FrankenSplit: Saliency Guided Neural Feature Compression with Shallow Variational Bottleneck Injection

1st Alireza Furutanpey
Distributed Systems Group
TU Vienna
Vienna, Austria
a.furutanpey@dsg.tuwien.ac.at

2nd Philipp Raith
Distributed Systems Group
TU Vienna
Vienna, Austria
p.raith@dsg.tuwien.ac.at

3rd Schahram Dustdar
Distributed Systems Group
TU Vienna
Vienna, Austria
dustdar@dsg.tuwien.ac.at

***Abstract*—Lightweight neural networks exchange fast inference for predictive strength. Conversely, large deep neural networks have lower prediction error but incur prolonged inference times and high energy consumption on resource-constrained devices. This trade-off is unacceptable for latency-sensitive and performance-critical applications. Offloading inference tasks to a server is unsatisfactory due to the inevitable network congestion by high-dimensional data competing for limited bandwidth and leaving valuable client-side resources idle. This work demonstrates why existing methods cannot adequately address the need for high-performance inference in mobile edge computing. Then, we show how to overcome current limitations by introducing a novel training method to reduce bandwidth consumption in Machine-to-Machine communication and a generalizable design heuristic for resource-conscious compression models. We extensively evaluate our proposed method against a wide range of baselines for latency and compressive strength in an environment with asymmetric resource distribution between edge devices and servers. Despite our edge-oriented lightweight encoder, our method achieves considerably better compression rates.**

***Index Terms*—Split Computing, Edge Computing, Edge Intelligence, Neural Compression, Feature Compression, Knowledge Distillation**

I. Introduction

Advancements in mobile edge computing (MEC) gave unprecedented growth in Machine-to-Machine (M2M) communication [8], enabling critical latency and performance-sensitive applications, such as disaster warning or cognitive assistance. However, the accelerating pervasiveness of mobile devices relying on high-dimensional data for visual applications has led to an insurmountable amount of network traffic exceeding all expectations. Numerous streams of visual sensor data competing for limited bandwidth will inevitably lead to network congestion, incurring intolerable response delays. Applying handcrafted image codecs such as PNG or WebP [25] before offloading inference tasks can mitigate network load. The caveat is that lossy codecs significantly drop prediction performance. Contrarily, lossless codecs cannot adequately reduce traffic due to limited potential for low bitrates. One solution is to forgo offloading entirely by onloading the inference task to the client. Although various solutions to execute lightweight DNNs exist [9], they are unacceptable where accurate inference is indispensable. Split Computing (SC) emerged as an alternative to the binary on or offload decision mechanism to address the needs of critical mobile applications. The basic idea is to split a neural network to process the shallow layers on constrained devices and send a reduced representation to the remaining deeper layers deployed on a server. SC is an attempt at combining the advantages of off and onloading by exploiting the available resources across the entire compute continuum (e.g., Edge, Fog, Cloud). SC is based on the same assumption as onloading, that mobile clients are increasingly equipped with energy-efficient AI accelerators capable of executing lightweight DNNs. However, current SC methods have limited applicability due to requiring a carefully calibrated runtime system or their specificity towards certain DNN architectures. Arguably, the potential of SC is inhibited by primarily focusing on shifting parts of the model execution from the server to the client. Due to fluctuating network conditions and the inherent resource asymmetry between the client and the server, finding the right balance is challenging, i.e., applying the same assumptions for onloading to SC is a fallacy. In other words, conceiving methods focusing on repurposing the mobile chips to execute shallow layers based on the assumption that they are suitable for lightweight DNNs ignores that server-grade hardware can execute the same layers by orders of magnitudes faster. Hence, the viability of SC methods 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 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?*

To this end, we draw from recent advancements in lossy learned image compression (LIC) [46]. Despite outperforming handcrafted codecs, LIC is unsuitable for real-time inference in MEC since they consist of large autoencoders and other complex mechanisms that are demanding even for server-grade hardware. Moreover, research in compression primarily

focuses on reconstruction for human perception containing information superfluous for M2M communication, i.e., we require novel methods tailored for feature compression. Feature compression has its roots in the information bottleneck (IB) principle [43] and provides the underlying objective to fit existing LIC training objectives for machine interpretability. Nevertheless, analogous to the deep variational IB (DVIB) [1], current neural feature compression methods typically place the bottleneck at the penultimate layer, i.e., a distributed DVIB assumes the client hosts almost the entire network. Contrastingly, the opposite is needed in MEC.

In this work, we coin the term Shallow Variational Information Bottleneck Injection (SVBI) to address the increasing network traffic by M2M communication for visual tasks. Specifically, we introduce *FrankenSplit*, a novel training and design heuristic for embeddable feature compression models. The design heuristic of FrankenSplit generalizes SC to arbitrary DNN architectures, such that it is not limited to a single encoder-decoder pair. Complementary, the training method demonstrates how allocating resources to neural compression instead of partially computing shallow layers alleviates the limitation of SC to rely on complex runtime systems and to utilize local resources only under specific conditions. Importantly, our encoder has only 0.14M with a compression rate that can provide state-of-the-art image classification performance in highly constrained environments with a data rate of 0.25 Mbps in around 0.15 seconds. For reference, MobileNetV2/V3 [16], [37] see widespread praise in MEC for their lightweight architecture with comparably mediocre accuracy despite having 25x-39x more parameters than our encoder.

We summarize the contributions of this work as:

- Thoroughly exploring how shallow and deep bottleneck injection differ for feature compression to identify the distinct challenges of the former.
- Introducing a novel saliency-guided training method to overcome the challenges of SVBI to train a lightweight encoder with limited capacity.
- Introducing an autoencoder design heuristic that permits attaching various decoders from arbitrary DNN architectures to one universal encoder.

Section II summarizes relevant work on SC and LIC. Section III describes the problem domain and progressively introduces the solution approach. Section III extensively justifies relevant performance indicators and evaluates several implementations of FrankenSplit against various baselines to assess our method's efficacy. Lastly, Section V facilitates future research directives by summarizing key insights and current limitations.

II. Related Work

A. Neural Data Compression

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 [13]. 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 [3].

Ballé et al. introduced the Factorized Prior (FP) entropy model and formulated the neural compression problem by finding a representation with minimal entropy [4]. An encoder network transforms the original input to a latent variable, capturing the input's statistical dependencies. The parametric entropy model is fitted towards the train data to approximate the unknown marginal distribution and remains fixed during inference. In follow-up work, Ballé et al. [5] and Minnen et al. [32] extend the FP entropy model with input adaptivity by including a hyperprior as side information for the prior. Minnen et al. [32] introduce the Joint Autoregressive and Hierarchical Priors (JAHP) entropy model, 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. The entropy models introduced by Ballé et al. and Minnen et al. are foundational for most recently published work on variational image compression.

2) Feature Compression: Feature compression has its roots in the (Deep Variational) Information Bottleneck principle [1], [43]. 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 [42]. Dubois et al. generalized the VIB for multiple downstream tasks and were the first to describe the feature compression task formally [10]. 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. [30], [31], 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 [4], [5]. Moreover, they do not analyze the intrinsic differences between feature and image compression nor explain their solution approach. 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. Additionally, we establish a link between the VIB principle and bottleneck injection in SC to draw insights from our results

and facilitate a promising research direction in MEC.

B. Split Computing

SC is a bandwidth and latency-aware deployment strategy for providing resource-constrained devices access to compute-intensive deep learning models. We distinguish between two orthogonal approaches to SC.

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 [17]. 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 input or feature tensor manipulation [2], [20], [21]. 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 old superseded DNNs with an early bottleneck [23].

2) Artificial Bottleneck Injection: Bottleneck Injection methods retain the simplicity of offloading without the need for a complex runtime system by shifting the effort towards modifying and re-training an existing base model (backbone) to replace the shallow layers with an artificial bottleneck. Eshratifar et al. replace the shallow layers of ResNet-50 with a deterministic autoencoder network [11]. A follow-up work by Jiawei Shao and Jun Zhang further considers noisy communication channels [41]. Matsubara et al. [29], and Sbai et al. [38] 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 [30] as the state-of-the-art for bottleneck injection.

Although bottleneck injection is promising, there are two problems with current methods. They typically rely on autoencoder for crude data compression or are intended for a specific class of neural network architecture, such as convolutional neural networks (CNNs).

This work addresses both limitations of naive bottleneck injection methods.

III. Problem Formulation & Proposed Method

The aim is to request real-time predictions from a large remote DNN while maximizing resource efficiency and minimizing bandwidth consumption. Figure 1 illustrates deployment strategies we will consider during evaluation. Notably, the receiver has significantly more resources than the sender. Strategy a) corresponds to existing offloading strategies with CPU-bound handcrafted codecs applied to the input and is undesirable since lossy codecs achieve low bitrates at a high predictive loss. In strategy b), we utilize the onboard accelerator and apply a LIC method. Although they achieve low bitrates at a considerably less predictive loss, they incur an excessive overhead. Strategy c) is our proposed method with an embeddable variational compression model. To overcome the limitations of a) and b), (i) we require a resource–conscious encoder that is maximally compressive about a useful representation without increasing the predictive loss. Additionally, (ii) the decoder should exploit the available server-side resources but still incur minimal overhead on the backbone. Lastly, (iii) a compression model should fit for different downstream tasks and architectural families (e.g., CNNs or Vision Transformers). Our solution approach focuses on two distinct but intertwined aspects. First is an appropriate training objective for feature compression with limited model capacity. The second concerns a practical implementation by introducing an architectural design heuristic for edge-oriented autoencoders.

A. Rate-Distortion Theory and the Information Bottleneck

By Shannon's rate-distortion theory [40], we seek a mapping from a random variable (r.v.) X to an r.v. Z, minimizing the bitrate the outcomes of X and can be reconstructed according to a distortion constraint D_c. More formally, given a distortion measure $\mathcal{D}$ and a distortion constraint D_c, the minimal bitrate is

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

where $I(X;Z)$ is the mutual information between the input and latent vector and is defined as

$$\mathrm{I}(X;Z) = \int\int p(x,z) \log \left(\frac{p(x,z)}{p(x)p(z)} \right) dxdz \tag{2}$$

In image compression, the distortion measure is between the original input x and distorted reconstruction $\tilde{x}$. Nevertheless, when data is only seen by machines, optimizing for perceptual quality results in Z to retain redundant information.

Consider the Data Processing Inequality (DPI). For any 3 r.v.s X, Y, Z that form a Markov chain $X \leftrightarrow Y \leftrightarrow Z$ the following holds:

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

Additionally, as argued by Tishby and Zaslavsky [44], we can view layered neural networks as a Markov chain of successive representations and describe the information flow in an n-layered sequential DNN, layer with the information path:

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

In other words, the final representation before a prediction R_n for a discriminative task cannot have more mutual information with the target than the input X and typically has less. Empirically, this is established by how slight perturbations and certain

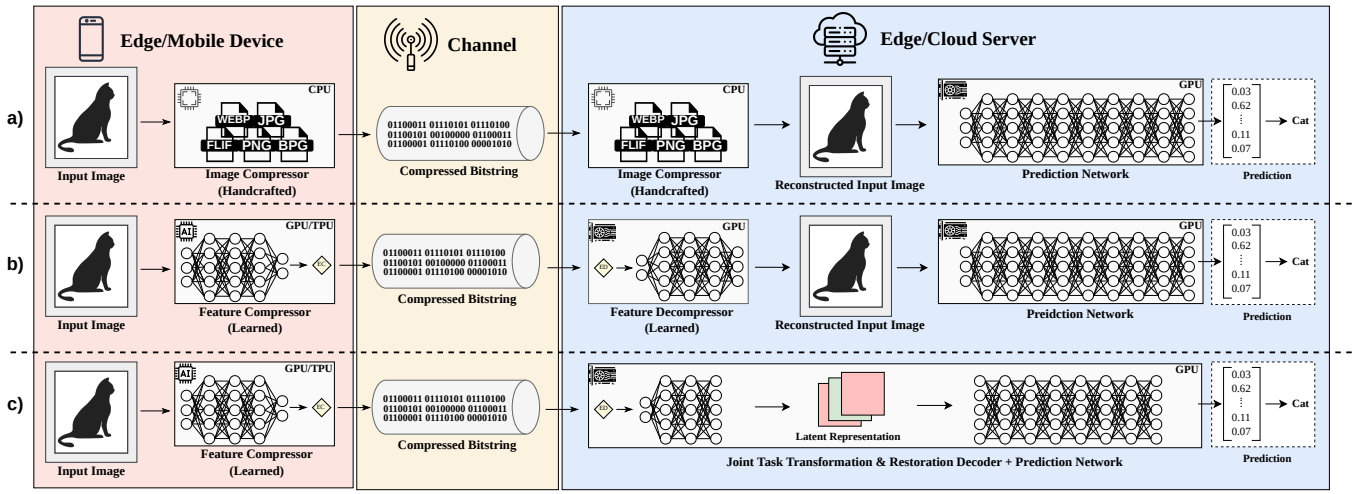

Fig. 1: Edge Inference Deployment Strategies

transformations do not affect predictions. Hence, when the task is to predict the ground-truth labels Y from a joint distribution $P_{X,Y}$, the rate-distortion objective is essentially given by the information bottleneck principle [43]. 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 is the conditional cross entropy (CE):

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

Unsurprisingly, using (6) for estimating the distortion measure in (5) for end-to-end optimization of a neural compression model is not a novel idea. However, a common assumption in such work is that the latent variable is the final representation R_n before performing the prediction, i.e., the encoder must consist of an entire backbone model 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)*. Figure 2 illustrates the concept of bottleneck injection. Intuitively, the existing methods for DVBI should generalize to SVBI should, e.g., approximating the distortion term with (6) as in [42]. Although by shifting the bottleneck to the shallow layers, the encoder has less capacity for finding a minimal sufficient representation, the objective is still an approximation of (1).

Head Distillation (HD) [29], [38] is an alternative method that naturally relates to shallow bottleneck. In HD, the removed head network teaches the artificial bottleneck. However, we will show that HD is a suboptimal approximation of (1) for

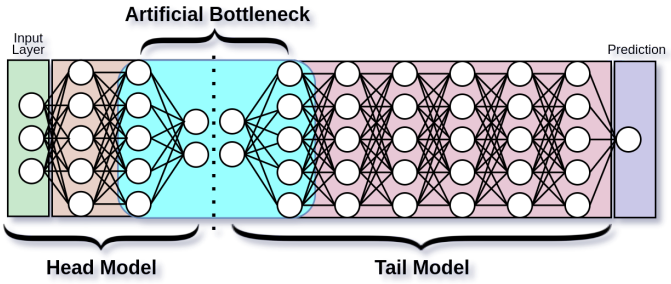

Fig. 2: Modifying an existing network

variational compression. In the following, we bridge the gap between deep and shallow bottleneck injection by formulating the VIB objective for HD.

B. (Head) Distilled Deep Variational Information Bottleneck

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 how SVBI naturally generalizes to multiple downstream tasks and DNNs to Section IV-C6.

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 $\mathcal{P}_{\mathcal{T}}(x) = \mathcal{P}_h(\mathcal{P}_t(x)) = \mathcal{P}_t(h)$. Lastly, replace the head with an autoencoder and 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 compression model. First, is *direct optimization* corresponding to the DVIB objective in (5), except we replace the CE with the standard KD loss [15] to approximate $I(Z;Y)$.

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 3 illustrates the difference between estimating the objectives (5) and (7). Intuitively, 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 holds, it does not optimize Z respective $Y_{\mathcal{T}}$. The marginal distribution of $Y_{\mathcal{T}}$ now arises from the r.v. X and the parameters of $\mathcal{P}_{\mathcal{T}}$. Moreover, since maximizing $I(Z;H)$ is only a proxy objective for $I(Z;Y_{\mathcal{T}})$, it corresponds to the Markov chain $Y_{\mathcal{T}} \leftrightarrow H \leftrightarrow X \leftrightarrow Z$. Notice how X is now conditional independent of $Y_{\mathcal{T}}$, breaking the assumption from the previous subsection that the original input is maximally informative of our target task. In other words, X is now conditionally independent of $Y_{\mathcal{T}}$, and the conditional mutual information $I(X;Y_{\mathcal{T}}|H)$ tells us by how much we "overshoot" in our estimation. Consequently, when training a compression model with (7), we skew the rate-distortion optimization towards a higher rate than necessary by setting the lower bound for the distortion estimation too high. Nevertheless, while Singh et al. demonstrated how direct optimization works well for DVBI [42], we show how it results in r-d performance poorer than handcrafted codecs for SVBI in Section IV. Interestingly, HD results in considerably better r-d performance despite its inherent suboptimality, which is why it is the focus of this work.

C. End-to-end Optimized Feature Compression for Partial Distilled Networks

Unlike most work on artificial bottleneck injection, our bottleneck is not a deterministic autoencoder. Instead, we embed a stochastic compression model. Categorically, we follow nonlinear transform coding [3] (NTC) to implement a neural compression algorithm. For an image vector x, we have a parametric analysis transform $g_a(x;\phi_g)$ maps x to a (partially) decorrelated latent vector z. Then, a quantizer Q discretizes z to $\bar{z}$, such that an entropy coder can losslessly compress $\bar{z}$ to a sequence of bits. Here, we start diverging from NTC for image compression. Rather than an approximate inverse of the analysis, the parametric synthesis transform $g_s(\bar{z};\theta_g)$ maps $\bar{z}$ to a representation $\tilde{h}$ that is suitable for various tail predictors and downstream tasks. The practical implementation of our objective resembles variational autoencoder-based image compression optimization, as introduced in [4], [5]. 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 $\tilde{z}$, which we assume is in a fully factorized form. Analogous to the LIC methods discussed in Section II-A, we apply uniform quantization $\bar{z} = \lfloor \tilde{z} \rceil$, which is essentially rounding to the nearest integer. Discretization leads to problems with the gradient flow. Hence, 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 approximating objective (7) as

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

Note, in the relaxed lagrangian of (1), the weight is on the distortion while we weight the rate term to align closer to the variational information bottleneck objective. Moreover, omitting the second term ($\beta = 0$) results in the head distillation objective for naive bottleneck injection.

Since the loss in (10) is an approximation of the objective (7), the model weights will not converge towards an optimum for objective (1). However, notice how the root of the suboptimality stems from treating every pixel of H equally crucial to the remaining layer. In the previous subsection, we established that as the remaining layers further process a representation h, its information content decreases monotonously. More intuitively, the shallow layers extract high-level features, while deeper layers are more focused. The implication here is that the MSE in (10) may overly strictly penalize pixels at spatial locations which contain redundant information that later layers can safely discard. Contrarily, the loss may not penalize the salient pixels strictly enough when $\tilde{h}$ is close to h numerically. This insight explains 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 sufficiently high. Since the representation of the last hidden layer and the representation of the shallow layer are so far apart in the information path, there is insufficient information to minimize $\mathcal{D}(H;\tilde{H})$. Effectively, the low information content $I(Y;\tilde{Y})$ explains why there is not enough signal to propagate down from the deeper layers.

Finally, to solve the suboptimality (10), assume we have access to a vector $\mathcal{S}$. Each $s_i \in \mathcal{S}$ is a weight term for a spatial location, which is salient about the conditional probability distributions of the remaining tail layers. Then, we should be

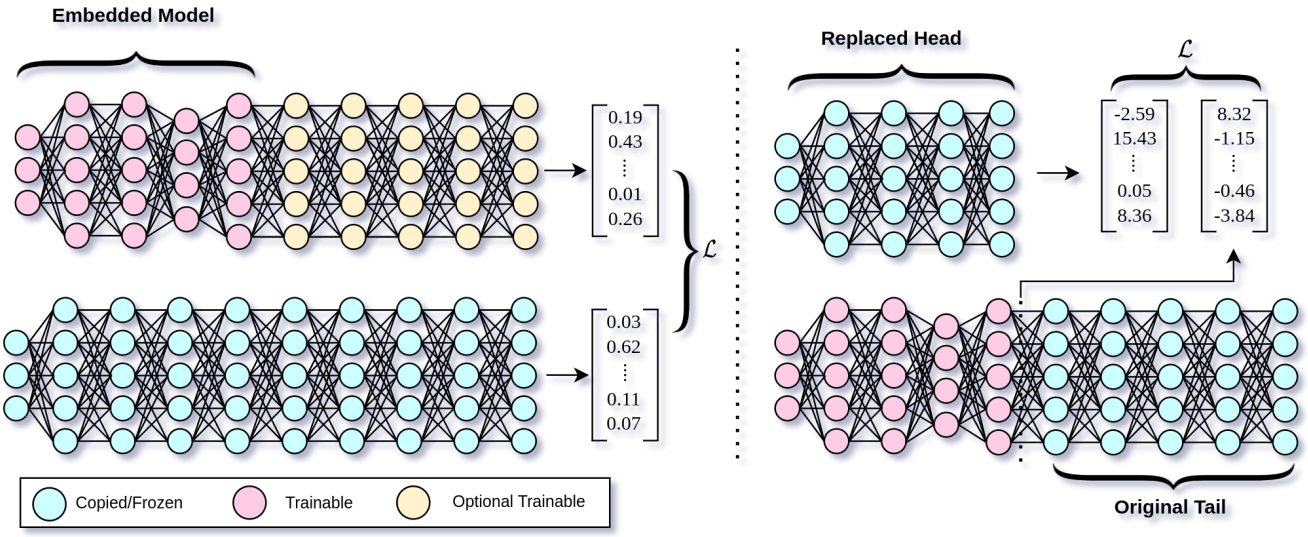

Fig. 3: In the left, the knowledge is transferred by the soft labels of the teacher. In the right, the replaced head directly transfers the knowledge to the embedded model

able to improve the rate-distortion performance by replacing the distortion term in (10) with

$$\mathcal{L}_{\text{distortion}} = \frac{1}{N}\sum_{i}(h_i - \tilde{h}_i)^2 \cdot s_i \tag{11}$$

D. Saliency Maps

Class Activation Mapping (CAM) is a method to increase the explainability of trained models [47]. Nevertheless, it effectively measures the importance of each spatial location. As illustrated in Figure 4, CAM allows us to extract attention maps from a pre-trained teacher model at any layer. The original literature extensively covers computing the saliency maps, so we do not cover the details. To derive vector $\mathcal{S}$ for

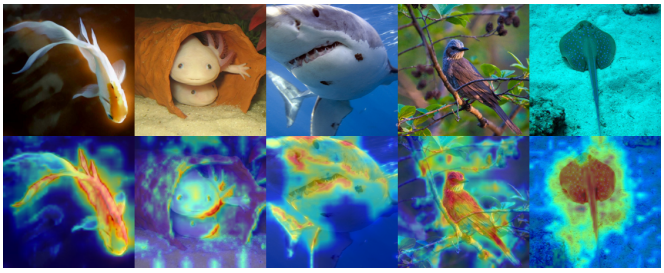

Fig. 4: Extracted Saliency Maps using Grad-CAM

the loss in (11), we use one of the Grad-CAM variants [39] to calculate and persist a saliency map for each sample as a preliminary step. Nevertheless, for data augmentation, it is preferable to calculate the maps ad-hoc, which is feasible depending on the CAM strategy. Since Grad-CAM allows us to derive pixel importance at any layer, each saliency map is the unweighted average of the maps created by the top-level layers.

E. 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 models. To not inflate the significance of our contribution, we refrain from including efficient neural network components based on existing work in efficient neural network design. A generalizable resource-asymmetry-aware autoencoder design remains; components should be reusable and attachable to arbitrary network architectures once trained.

When performing predictions with the latent, it may seem redundant to have a synthesis transform. Since we cannot increase the information content of Z, we could directly feed the encoder output to the tail of a backbone model. However, this locks the encoder to only a single backbone. The challenge is to conceive a design heuristic that can exploit the available server resources to aid the lightweight encoder with minimal overhead on the prediction task.

1) Inter- and Intra-architecture module re-attachment: We assume that, after training an embedded compression model with one particular backbone as described in Section III-C, it is possible to reuse the encoder or decoder for different backbones with little effort based on the following two observations.

First is how most modern DNNs consist of an initial embedding followed by a few top-level layers. The top-level layers consist of coarse-grained layers, while the coarse-grained ones recursively consist of finer-grained ones. The terminology differs for each work, but here we will refer to top-level layers as *stages* and the coarse-grained layers as *blocks*. Within architectural families, the individual components are identical. The difference between variants is primarily the embed dimen-

sions or the block ratio of the deepest stages. For example, the block ratio of ResNet-50/Swin-T is 3:4:**6**:3/2:2:**6**.2, while the block ratio of ResNet-101/Swin-S is 3:4:**23**:3/2:2:**18**:2. Importantly, stages define the shallow layers (head model) as all the layers before the deepest stage. Additionally, there is a natural interface for the decoder to attach to multiple backbones with minimal effort, which we refer to as *intra-architecture module re-attachment*.

This second observation is how network families introduce different inductive biases (e.g., convolutions versus attention modules with sliding windows), leading to diverging representations among non-related architectures. Therefore, it is naive to disregard architecture-induced bias by directly repurposing neural compression models for SC. For example, Matsubara et al. [31] use a scaled-down version of Ballé et al.'s [4] convolutional neural compression model, demonstrating strong rate-distortion performance for bottlenecks reconstructing a convolutional layer. However, we will show that this does not generalize to other architectural families, such as hierarchical vision transformers [24].

One potential solution is to leverage identical components from a target network. While this approach may be inconsequential for decoders, given the homogeneity of server hardware, 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. Consequently, regardless of the encoder or decoder, we account for the heterogeneity with a universal encoder composed of three downsampling residual blocks of two stacked 3×3 convolutions, totaling around 140k parameters.

Since the mutual information between the original input and shallow layers is still high, it should be possible to add a that maps the representation of the encoder to one suitable for different architectures based on invariance towards invertible transformations of mutual information. Specifically, for invertible functions ϕ, ψ, it holds that:

$$I(X;Y) = I(\phi(X);\psi(Y)) \tag{12}$$

We refer to re-mapping a representation from one family to another as *inter-architecture module re-attachment*.

2) Resource-Asymmetry Aware Network Design: The number of autoencoder parameters is typically comparable between the encoder and decoder. Increasing the decoder parameters does not change the inability to recover lost signals (Section III-A), i.e., it does not offset the limited encoder capacity for accurately distinguishing between valuable and redundant information. Nevertheless, when optimizing the entire compression pipeline simultaneously, the training algorithm should consider the ability for restoration by the decoder, which in turn should help the encoder parameters to converge towards finding a minimal sufficient statistic respective to the task. Hence, Inspired by [22], we partially treat decoding as an *image restoration problem* and add restoration blocks with optional upsampling between backbone-specific transformation blocks. The incurred latency is minimal and is offset by skipping the shallow layers.

To summarize, our network architecture is a general design heuristic, i.e., the individual components are interchangeable with the current state-of-the-art building blocks for vision models. Figure 5 illustrates the purpose of the decoder in restoring and transforming the latent. Moreover, it captures the essence of FrankenSplit as the re-usability through (re-) attaching arbitrary networks regardless of their architecture and target task. Due to space constraints, the figure does not

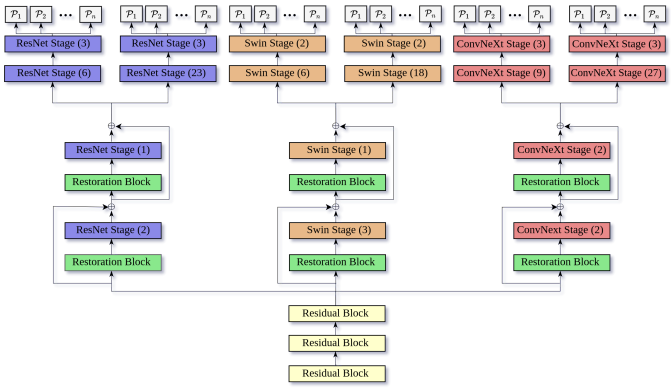

Fig. 5: Multiple decoders spanning from a single encoder and multiple tails attached to a single decoder. The number in parenthesis represents stage depth.

accurately represent the exact architectures of the models, and we cannot describe each decoder architecture in detail. Instead, the accompanying repository contains descriptive configurations.

IV. Evaluation

A. Experiment Setting

The experiments reflect the deployment strategies illustrated in Figure 1. Ultimately, we aim to demonstrate that FrankenSplit enables latency-sensitive and high-performance 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 inference 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., to not limit FrankenSplit to specific network conditions, it must remain competitive even when stronger connections are available (e.g., 4G LTE)

Lastly, FrankenSplit should still be applicable as newer and more powerful DNN predictors emerge, i.e., it is vital to evaluate 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 [14] for classic residual CNNs.

(ii) Swin Transformer [24] for hierarchical vision transformers, which are receiving increasing adaptation for a wide variety of vision tasks. (iii) ConvNeXt for modernized state-of-the-art CNNs. Table I summarizes the relevant characteristics of the unmodified backbones subject to our experiments.

TABLE I: Overview of Used Backbone Models on the Server

Backbone	Stage 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

1) Baselines: Since our work aligns closest to learned image compression, we extensively compare FrankenSplit with learned and 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 hyper networks for 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 Balle et al. [4], [5] and Minnen et al. [32] for LIC methods since they represent foundational milestones. Complementary, we include the work by Cheng et al. 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. 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. However, 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. [26] 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. Nevertheless, according to their results, we can assume a near-identical r-d performance as the work by Cheng et al.

As a baseline for the state-of-the-art SC, we include the Entropic Student (ES) [30], [31]. Crucially, the ES demonstrates the performance of naively applying a minimally adjusted LIC method for feature compression without considering the intrinsic properties of the problem domain we have derived in Section III-A. 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. However, comparing bottleneck injection methods using different backbones is fair, as we found that the choice does not significantly impact the r-d performance. 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 abovementioned problems.

2) Criteria rate-distortion performance: We primarily measure the bitrate in bits per pixel (bpp) which is sensible 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, adhering to the customs of SC by reporting absolute values, such as top-1 accuracy, gives an unfair advantage to experiments conducted on higher capacity backbones and veils the efficacy of a proposed method. Consequently, we evaluate the distortion with the relative measure *predictive loss*. To ensure a fair comparison, we give the LIC and handcrafted baselines a grace threshold of 1.0% top-1 accuracy, considering it is possible to mitigate some predictive loss incurred by codec artifacts [27]. For example, if a model has a predictive loss of 1.5% top-1 accuracy, we report it as 0.5% below our definition of lossless prediction. However, for FrankenSplit, we set the grace threshold at 0.4%, reflecting the configuration with the lowest predictive loss of the ES. Note that the ES improves r-d performance by finetuning the backbone weights by applying a secondary training stage. In contrast, we put our work at a disadvantage by training, FrankenSplit exclusively with the methods introduced in this work.

3) Measuring latency and overhead: The second category concerns with execution times of prediction pipelines with various wireless connections. To account for the resource asymmetry in MEC, we use NVIDIA Jetson boards to represent the capable but resource-constrained mobile client, and the server hosts a powerful GPU. Table II summarizes the hardware we use in our experiments.

TABLE II: 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

B. Training & Implementation Details

We optimize our compression models initially on the 1.28 million ImageNet [36] training samples for 15 epochs, according to the strategies described in section III-C and section III-D, with some slight practical modifications for stable training. Importantly, FrankenSplit's HD training method is different from the baseline ES. The former exclusively uses the MSE between the synthesis transform and the pruned

backbone head, while the latter additionally includes the MSE between the remaining tail stages, i.e., the ES training method does not faithfully implement HD. We found including the MSE between the deeper layers leads to unstable and considerably prolonged training. However, the training hyperparameters are similar enough to the ES to ensure that rate-distortion improvements are due to the methods introduced in this work. Concretely, we use Adam optimization [18] with a batch size of 16 and start with an initial learning rate of $1{\cdot}10^{-3}$, then gradually lower it to $1{\times}10^{-6}$ with an exponential scheduler.

An essential hyperparameter is β which controls the rate term during training, i.e., higher β trades lower bitrates with higher distortion. We aim to minimize bitrate without sacrificing predictive strength. Hence, we first seek the lowest β resulting in lossless prediction. For the lossless configuration, we ensure that it consistently results in lossless prediction and train models five times.

For verification, implementations of FrankenSplit, including experiment configurations, are publicly available[1]. We use PyTorch [35] to implement our models, CompressAI [6] for entropy estimation and entropy coding, and pre-trained backbones from PyTorch Image Models [45]. All baseline implementations and weights were either taken from CompressAI or the accompanying repository of a baseline. To compute the saliency maps, we use a modified *XGradCAM* method from the library in [12] and include necessary patches in our repository. Lastly, to ensure reproducibility, we use torchdistill [28].

C. Rate-Distortion Performance

We measure the predictive loss by the drop in top-1 accuracy using the ImageNet validation set for the standard classification task with 1000 categories. Analogously, we measure physical filesizes of the entropy-coded binaries to calculate the average bpp. Figure 6 shows rate-distortion curves with the Swin-B backbone. The architecture of FrankenSplit-FP and FrankenSplit-SGFP are identical and based on Section III-E. We train both models with the loss functions derived in Section III-C. The difference is that FrankenSplit-SGFP is saliency guided, i.e., FrankenSplit-FP represents the suboptimal HD training method and is an ablation to our proposed solution.

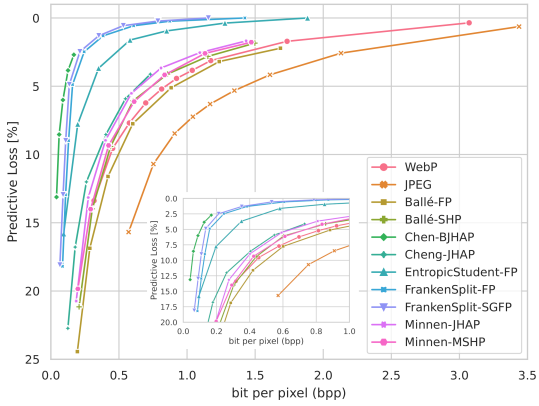

Fig. 6: Rate-distortion curve for ImageNet

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

1) Effect of Saliency Guidance: Although FrankenSplit-FP performs better than almost all other models, it is trained with the suboptimal objective discussed in Section III-B. Empirically, this is demonstrated by FrankenSplit-SGFP outperforming FrankenSplit-FP on the r-d curve. By simply guiding the distortion loss with saliency maps, we achieve a 25% lower bitrate at no additional cots.

2) Comparison to the Entropic Student: Even without saliency guidance, FrankenSplit-FP consistently outperforms the ES by a large margin. Specifically, FrankenSplit-FP and FrankenSplitSG-FP achieve 32% and 63% lower bitrates for the lossless configuration.

For a direct comparison to the ES, we ensured that our bottleneck injection incurs comparable overhead. Moreover, the ES has an advantage due to fine-tuning tail parameters in an auxiliary training stage. Therefore, we attribute the performance gain to the more sophisticated architectural design decisions described in Section III-E.

3) Comparison to Image Codecs: For almost all lossy codec baselines, Figure 6 illustrates that FrankenSplit-(SG)FP has a significantly better r-d performance. Since we refrain from using hyper networks for side information or a context model, comparing FrankenSplit-FP to Ballé-FP, best 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 FrankenSplit-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 complex or heavy-weight mechanisms. The only baseline, FrankenSplit does not convincingly outperform is Chen-BJHAP. Nevertheless, incurred overhead is as vital for real-time inference, which we will evaluate Section IV-D.

4) Naivety of Direct Optimization for SVBI: In Section III-C we mentioned that direct optimization does not work for SVBI as it does for DVBI, where the bottleneck is at the penultimate layer. Interestingly, direct optimization performs incomparably worse than HD, despite the latter's inherent suboptimality. Moreover, we identified the insufficient mutual information $I(Y, \tilde{Y})$ as the cause.

Contrasting the r-d performance on the simple CIFAR-10 [19] dataset summarized in the left with the r-d performance on the ImageNet dataset on the right in Figure 7 provides empirical evidence. Other than training direct optimization methods for more epochs to account for slower convergence, all models are identical and optimized with the setup described in Section IV-B.

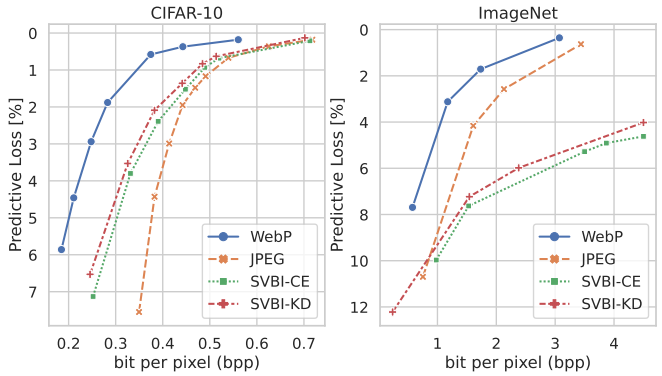

Fig. 7: Contrasting the r-d performance

On the trivial task, SVBI-CE and SVBI-KD yield moderate performance gain over JPEG. Contrarily, the same training method entirely falters on the ImageNet dataset and does not get close to lossless prediction even at high bitrates. When the joint distribution $P_{X,Y}$ arises from a trivial dataset, $I(Y;\tilde{Y})$ has just enough information to propagate down for the compression model to come close to a sufficient statistic of Z respective Y. However, when $P_{X,Y}$ arises from a non-trivial dataset, where the information content of $I(Y;\tilde{Y})$ is not enough to approximate a sufficient statistic, it performs significantly worse than both handcrafted codecs.

5) Generalization to arbitrary backbones: In the previous, we intentionally demonstrated the r-d performance of FrankenSplit with a Swin backbone since it implicitly backs up our claim from Section III-E that the synthesis transform can map the CNN encoder output to the input of a hierarchical vision transformer. Nevertheless, we initially set out to plot the r-d curves for all the models summarized in Table I but found that the choice of backbone has a negligible impact on the rate-distortion performance, i.e., with suitable hyperparameters, we get comparable results with FrankenSplit. Additionally, due to the relative distortion measure, different backbones will not lead to significantly different r-d curves for the baseline codecs. The insignificance of teacher choice for r-d performance is consistent with all our claims and findings. With the objective in (7), we learn the representation of a shallow layer, which generalizes well since such representations typically have high mutual information with the original input. Moreover, the difference between the smaller and larger model variations from the same architectural family is primarily due to the deepest stage or the embed size. Likewise, plotting results for intra-architecture module re-attachment is redundant, as it will result in nearly identical r-d curves after finetuning the decoder for a few epochs. What remains is to demonstrate the necessity of our heuristic regarding inductive bias, i.e., why current bottleneck injection approaches do not generalize beyond their intended target architectural family.

As detailed in Section III-E, the encoder architecture is the same regardless of the backbone. In contrast, the decoder consists of interchangeable restoration and transformation blocks. Irrespective of the backbone, restoration blocks are residual sub-pixel or transpose convolutions. Contrarily, the transformation block depends on the target backbone, i.e., we argued that the r-d performance improves when the transformation block induces the same bias as the target backbone. Specifically, we have a synthesis transform blueprint for each of the three architectural families (Swin, ResNet, and ConvNeXt) that results in comparable rate-distortion performance from Figure 6 for their intended variations. For example, for any ConvNeXt backbone variation, we train the bottleneck with the ConvNeXt synthesis blueprint analogous to how we used the Swin transformer blueprint for the results from earlier. Table III summarizes the r-d performance of using the ConvNeXt and ResNet blueprint on the Swin-B backbone. Once we attempt to use the ResNet or ConvNeXt restoration blocks, the rate-distortion performance for the Swin-B is significantly worse. Most notably, the lossless configuration for Swin-Swin has less than half the average file size of ConvNeXt-Swin with 2.49% predictive loss.

TABLE III: Applying correct and false blueprints to a Swin backbone

Blueprint-Backbone	File Size (kB)	Predictive Loss (%)
Swin-Swin	8.70	0.00
	5.08	0.40
	3.19	0.77
ConvNext-Swin	19.05	2.49
	15.07	3.00
	10.05	3.35
ResNet-Swin	22.54	0.82
	18.19	0.99
	8.27	1.34

Intra-model re-attachment and inter-model re-attachment by training a new encoder agnostic towards the decoder architecture works as hypothesized. Still, our results found that inter-module architecture incurs a considerable predictive loss if we introduce a freshly initialized decoder attached to a pre-trained encoder. However, in the following, we demonstrate that, as long as the backbone is from the same family, FrankenSplit generalizes to multiple tasks without re-training any components.

6) Generalization to multiple Downstream Tasks: Dubois et al. [10] generalize a bottleneck to multiple tasks by seeking a maximal invariant through extensive augmentation and contrastive SSL [34]. Here we argue that our SVBI naturally generalizes despite its simpler training method involving no augmentation or contrastive SSL for two reasons. First, the head distillation method optimizes our encoder to learn high-level representations, i.e., the mutual information between the bottleneck output and the original input is still high. Additionally, hierarchical vision models are primarily feature extractors. Since the bottleneck learns to approximate the variational density from a challenging distribution, it should be possible to embed the same compression module on backbones trained for other tasks. We provide empirical evidence for this claim by finetuning a predictor prepared on ImageNet for different classification tasks.

For FrankenSplit, 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 [7] dataset since it contains noise in high pixel intensities. Additionally, we include CIFAR-100 [19]. Lastly, to contrast more challenging tasks, we include Flower-102 [33] datasets. Besides the relatively low number of samples per class, the Flower-101 dataset has no inherent difficulty.

We freeze the entire compression model trained on the ImageNet dataset and finetune the tails separately for five epochs with no augmentation, a learning rate of $5 \cdot 10^{-5}$ using Adam optimization. The teacher backbones achieve an 87.73%, 88.01%, and 89.00% top-1 accuracy, respectively. Figure 8 summarizes the r-d performance for each task. Note that the focus of this study is not transfer learning but to show how the bottleneck retains information necessary to accommodate multiple predictors. Our method still demonstrates clear r-d performance gains over the baselines. More importantly, notice how FrankenSplit-SGFP outperforms FrankenSplit-FP on the r-d curve for the Food-101 dataset, with a comparable margin to the ImageNet dataset. Contrarily, on the Flower-102 datasets, there is noticeably less performance difference between them. Presumably, on trivial datasets, the suboptimality of HD is less significant. Considering how easier tasks require less model capacity, the diminishing efficacy of our saliency-guided training method is consistent with our claims and derivations in Section III. The information of the shallow layer may suffice, such that the residual connections skip most of the deeper layers. Simply put, the less necessary the activations of the deeper layers are, the better a minimal sufficient statistic respective H approximates a minimal sufficient statistic respective Y.

D. Prediction Latency and Overhead

We report computational overhead by latency in ms rather than throughput due to measuring the individual components in our distributed inference pipeline. Moreover, we omit the baseline entropic student, as it does not generalize to all backbones we consider. Still, since we designed our autoencoder to have a similar model size, the computational overhead is comparable when FrankenSplit's backbone has a ResNet-like architecture. Additionally, we exclude entropy coding from our measurement since not all baselines use the same entropy coder, and their implementation is far from optimal concerning execution time. Lastly, the results implicitly assume the Swin-B backbone for the remainder of this section due to space constraints. Inference times with other tails for FrankenSplit can be derived fromTable IV. Analogously, the inference times of applying LIC models for different unmodified backbones can be derived using Table I. Notably, the relative overhead decreases the larger the tail is, which is favorable since we target inference from more accurate predictors.

1) Computational Overhead: We first disregard network conditions to get an overview of the computational overhead of applying compression models. Table V 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.

TABLE IV: Execution times 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

TABLE V: 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/ 351M	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

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. Comparing Ballé-FP to Minnen-MSHP reveals that including side information only incurs minimal overhead, even on the constrained client. Contrarily, Lu-JHAP has less than half the parameters of Minnen-MSHP but is still slower by two orders of magnitude. On the server, FrankenSplit's is slightly slower than the baselines without a context model due to the attention mechanism of the decoder blueprint for the swin backbone. However, unlike all other baselines, despite the more extensive decoder network, the computational load is near evenly distributed between the client and the server, i.e., FrankenSplit's design heuristic successfully considers resource asymmetry. The importance is emphasized by how the partially parallelized context model of Chen-BHJAP 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.

2) Competing against Offloading: In Section I, we argued the importance of exploiting available resources for resource efficiency. Although SC attempts resource efficiency with a

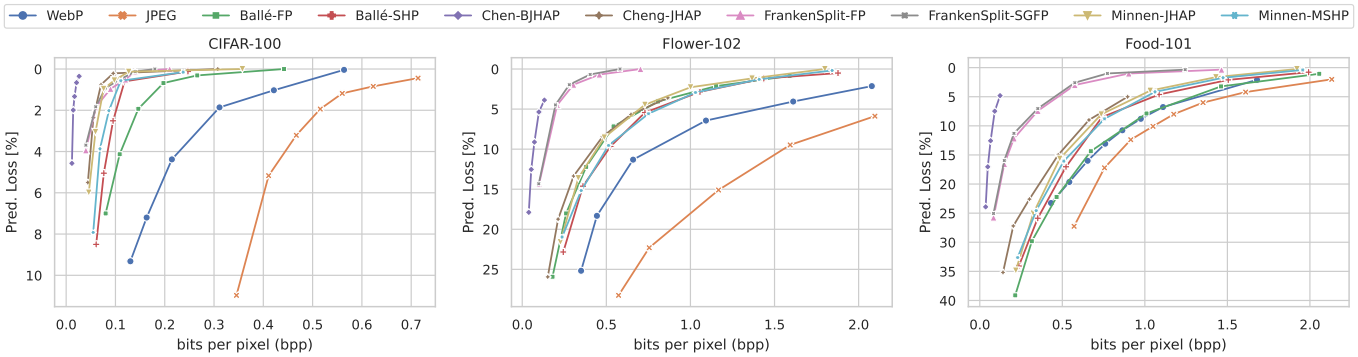

Fig. 8: Rate-distortion curve for multiple downstream tasks

joint on- and offloading deployment strategy, current methods are limited to niche applications. To overcome the niche applicability, we proposed treating compression as the primary objective and skipping the shallow layers on the server as a side product. Notably, due to tasking the decoder with restoring and transforming the compressed representation, the reduced latency by pruning the head is offset by the decoding time, i.e., unlike other work in SC, FrankenSplit does not reduce the latency on the server. Indicatively, FrankenSplit skips the head of the patch embed and the first two stages, but the decoding time offsets latency reduction. Hence, latency reduction exclusively depends on the reduced transfer size.

We evaluate transfer time on a broad range of standards using the average compressed filesize from the ImageNet validation as the transfer size. We set the encoding and decoding time for the handcrafted codecs to 0 since we did not include the execution time of entropy coding for learned methods. The setting favors the baselines because both rely on entropy coding and sequential CPU-bound transforms. Table VI summarizes how our method compares to the baselines in various standards, ranging from LoRa (0.05 Mbps) to 5G (66.9 Mbps). Due to space constraints, we cannot include comparisons to LIC models. Still, with Table V and the r-d results from the previous subsection, it is straightforward to infer that the LIC baselines have considerably higher latency than FrankenSplit. Instead, a comparison to the lightweight handcrafted codecs is more insightful.

Our method is virtually always preferable since the reduced transfer size is up to 12.06x smaller than the handcrafted codecs. Moreover, in highly constrained networks, such as LoRa, FS-SGFP (-0.23%) is up to 17.4x faster. For stronger connections, such as 4G LTE, it is still up to 3.3x faster. Generally, the less constrained the bandwidth, the more favorable it is to offload. Yet, FrankenSplit is significantly better than offloading up until 5G. The only caveat is that our method assumes the client has an onboard AI accelerator. However, since such chips already see widespread adoption and are becoming increasingly energy efficient, the assumption is sensible.

TABLE VI: Total Latency with various wireless standards

Standard/ Data Rate (Mbps)	model	Transfer (ms)	Total [TX2] (ms)	Total [NX] (ms)
LoRa/ 0.05	FS-SGFP (0.23)	769.98	787.87	785.92
	FS-SGFP (LL)	1133.41	1151.3	1149.35
	WebP	4675.95	4689.1	4689.1
	PNG	13675.92	13689.07	13689.07
BLE/ 0.27	FS-SGFP (0.23)	142.59	160.48	158.53
	FS-SGFP (LL)	209.89	227.78	225.83
	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
	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
	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
	WebP	3.49	16.64	16.64
	FS-SGFP (LL)	0.85	18.73	16.78
	PNG	10.22	23.36	23.36

V. Conclusion

In this work, we have carefully identified the unique challenges for critical applications relying on large DNNs, that edge devices cannot accommodate. Then, we analyzed the inadequacy of existing solutions for MEC, allowing us to conceive a novel training method and a design heuristic for lightweight autoencoders. We thoroughly evaluated our solution with various baselines and successfully demonstrated how it overcomes current limitations. Our evaluation has also led to two crucial insights. First, applying existing LIC methods to feature compression directly is naive. Second, including side information is less resource intensive than initially assumed. Therefore, it is interesting to conceive novel methods for feature compression to include side information beyond applying the hypernetworks from general-purpose LIC.

Acknowledgment

We sincerely thank Alexander Knoll for his dedication to keeping up our hardware running in the midst of chaos.

REFERENCES

[1] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. *CoRR*, abs/1612.00410, 2016.

[2] Mario Almeida, Stefanos Laskaridis, Stylianos I Venieris, Ilias Leontiadis, and Nicholas D Lane. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems*, 21(6):1–24, 2022.

[3] Johannes Ballé, Philip A. Chou, David Minnen, Saurabh Singh, Nick Johnston, Eirikur Agustsson, Sung Jin Hwang, and George Toderici. Nonlinear transform coding. *CoRR*, abs/2007.03034, 2020.

[4] Johannes Ballé, Valero Laparra, and Eero 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.

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior, 2018.

[6] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

[7] Lukas Bossard, Matthieu Guillaumin, and Luc 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*, pages 446–461. Springer, 2014.

[8] U Cisco. Cisco annual internet report (2018–2023) white paper. *Cisco: San Jose, CA, USA*, 10(1):1–35, 2020.

[9] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[10] Yann Dubois, Benjamin Bloem-Reddy, Karen Ullrich, and Chris J Maddison. Lossy compression for lossless prediction. *Advances in Neural Information Processing Systems*, 34:14014–14028, 2021.

[11] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud 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)*, pages 1–6, 2019.

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

[13] V.K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5):9–21, 2001.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[17] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas 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*, pages 1–15, 2020.

[21] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu 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)*, pages 671–678. IEEE, 2018.

[22] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1833–1844, 2021.

[23] Hongzhou Liu, Wenli Zheng, Li Li, and Minyi Guo. Loadpart: Load-aware dynamic partition of deep neural networks for edge offloading. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 481–491, 2022.

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[25] Google LLC. An image format for the web.

[26] Ming Lu, Peiyao Guo, Huiqing Shi, Chuntong Cao, and Zhan Ma. Transformer-based image compression. In *2022 Data Compression Conference (DCC)*, pages 469–469, 2022.

[27] Xiyang Luo, Hossein Talebi, Feng Yang, Michael Elad, and Peyman Milanfar. The rate-distortion-accuracy tradeoff: Jpeg case study. *arXiv preprint arXiv:2008.00605*, 2020.

[28] Yoshitomo Matsubara. torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 24–44. Springer, 2021.

[29] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer 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, page 21–26, New York, NY, USA, 2019. Association for Computing Machinery.

[30] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. Sc2: Supervised compression for split computing, 2022.

[31] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2685–2695, 2022.

[32] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression, 2018.

[33] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008.

[34] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[35] Automatic Differentiation In Pytorch. Pytorch, 2018.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[37] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[38] Marion Sbai, Muhamad Risqi U. Saputra, Niki Trigoni, and Andrew Markham. Cut, distil and encode (cde): Split cloud-edge deep inference. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9, 2021.

[39] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019.

[40] Claude E Shannon. Coding theorems for a discrete source with a fidelity criterion. In *IRE National Convention Record, 1959*, volume 4, pages 142–163, 1959.

[41] Jiawei Shao and Jun 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)*, pages 1–6, 2020.

[42] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Johannes Ballé, Abhinav Shrivastava, and George Toderici. End-to-end learning of compressible features. *CoRR*, abs/2007.11797, 2020.

[43] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000.

[44] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pages 1–5. IEEE, 2015.

[45] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.
[46] Yibo Yang, Stephan Mandt, and Lucas Theis. An introduction to neural data compression, 2022.
[47] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.