Neural Network Coding of Difference Updates for Efficient Distributed Learning Communication

Daniel Becking, *Graduate Student Member, IEEE*, Karsten Müller, *Senior Member, IEEE*, Paul Haase, Heiner Kirchhoffer, Gerhard Tech, Wojciech Samek, *Member, IEEE*, Heiko Schwarz, Detlev Marpe, *Fellow, IEEE*, and Thomas Wiegand, *Fellow, IEEE*

***Abstract*—Distributed learning requires a frequent communication of neural network update data. For this, we present a set of new compression tools, jointly called differential neural network coding (dNNC). dNNC is specifically tailored to efficiently code incremental neural network updates and includes tools for federated BatchNorm folding (FedBNF), structured and unstructured sparsification, tensor row skipping, quantization optimization and temporal adaptation for improved context-adaptive binary arithmetic coding (CABAC). Furthermore, dNNC provides a new parameter update tree (PUT) mechanism, which allows to identify updates for different neural network parameter sub-sets and their relationship in synchronous and asynchronous neural network communication scenarios. Most of these tools have been included into the standardization process of the NNC standard (ISO/IEC 15938-17) edition 2. We benchmark dNNC in multiple federated and split learning scenarios using a variety of NN models and data including vision transformers and large-scale ImageNet experiments: It achieves compression efficiencies of 60% in comparison to the NNC standard edition 1 for transparent coding cases, i.e., without degrading the inference or training performance. This corresponds to a reduction in the size of the NN updates to less than 1% of their original size. Moreover, dNNC reduces the overall energy consumption required for communication in federated learning systems by up to 94%.**

***Index Terms*—Neural network coding, federated learning, transfer learning, split learning, efficient NN communication, ISO/IEC MPEG standards, federated batchnorm folding.**

I. Introduction

NEURAL network compression has received increasing attention in recent years and led to the specification of the international NNC standard ISO/IEC 15938-17 [1] in 2022 for compressing full or "base" neural networks (NNs). This development has been driven by a number of factors: First, more and more applications in a variety of fields [2], [3], [4], [5], [6] utilize machine learning and artificial intelligence. Second, the underlying NNs became more and more complex with increasing structure complexity, number of layers and number of parameters per layer, such that current NNs have billions of parameters. And third, new distributed use cases have emerged, where many devices share the training of NNs. One application of this use case is federated learning (FL) [7], where a common NN is trained on multiple client devices, each using a training data subset. The training is orchestrated by a server, which aggregates the trained NN variants (e.g., by averaging). Another variant is federated distillation [8], where only soft labels, i.e., NN output data is exchanged. Here, clients may even train different NN architectures, as long as the last layer structure is identical. Another application is split learning (SL) [9], where NNs are split among devices, e.g., one device trains the first k layers, while another device trains the remaining layers.

Federated learning systems are often used in cross-device or cross-silo configurations with different communication characteristics: Cross-silo systems mostly involve a number of active, stable devices. For FL, this results in synchronous communication, i.e., all clients train and provide updates at all communication rounds. In contrast, cross-device systems involve much more versatile, distributed and often mobile devices. This results in asynchronous FL communication with client devices being inactive for a number of rounds.

Despite the benefits of distributed learning (e.g., basic data protection due to keeping training data on local client devices), it also encounters some challenges. Since NN data is frequently exchanged between devices, the learning process is constrained by the limited bandwidth of the communication channel. Consequently, adequate compression to reduce the large amount of communicated data might be required. Client drift is also a major issue, as heterogeneities in training data, model architecture, or the devices' computational resources can vary significantly, resulting in an overall difficult training process. Robustness against adversarial devices and privacy preservation are additional challenges in distributed learning and subject of current research [10].

This paper focuses primarily on the communications cost challenge and also aims at potential energy savings due to compressed communication. In general, research on

Manuscript received 20 June 2023; revised 8 October 2023 and 12 January 2024; accepted 18 January 2024. Date of publication 23 January 2024; date of current version 24 July 2024. The Associate Editor coordinating the review of this manuscript and approving it for publication was Prof. R. Hong. *(Corresponding authors: Daniel Becking; Karsten Müller.)*

Daniel Becking, Karsten Müller, Paul Haase, Heiner Kirchhoffer, Gerhard Tech, and Detlev Marpe are with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institute, 10587 Berlin, Germany (e-mail: daniel.becking@hhi.fraunhofer.de; karsten.mueller@hhi.fraunhofer.de; paul.haase@hhi.fraunhofer.de; heiner.kirchhoffer@hhi.fraunhofer.de; gerhard.tech@hhi.fraunhofer.de; detlev.marpe@hhi.fraunhofer.de).

Wojciech Samek and Thomas Wiegand are with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institute, 10587 Berlin, Germany, and also with Technical University of Berlin, 10623, Berlin, Germany (e-mail: wojciech.samek@hhi.fraunhofer.de; thomas.wiegand@hhi.fraunhofer.de).

Heiko Schwarz is with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institute, 10587 Berlin, Germany, and also with the Free University of Berlin, 14195 Berlin, Germany (e-mail: heiko.schwarz@hhi.fraunhofer.de).

This article has supplementary downloadable material available at https://doi.org/10.1109/TMM.2024.3357198, provided by the authors.

Digital Object Identifier 10.1109/TMM.2024.3357198

communication-efficient FL may be classified into two main categories: Reducing the total number of bits transferred per communication round and reducing the total number of communication rounds, thus promoting local learning more. Both can cause significant performance degradation and slow down training. In summary, current developments of distributed NN systems require high data compression without degradation of inference quality or convergence speed, flexible communication, and also a reasonable benchmarking.

The paper is organized as follows: Section II presents related work and our contributions. Section III describes communication in federated learning systems and the handling of sent and received (difference) NNs. Section IV describes the proposed difference NN coding (dNNC) method with new coding tools. In Section V, experimental results and applications are discussed and finally, the paper is concluded in Section VI.

II. Related Works and Our Contribution

Distributed learning methods like those in [11], [12] preceded Federated Learning, primarily in synchronized data center setups with hardwired machines. Asynchronous strategies were explored in [13], while [14] extended this to deeper NNs, sharing parameter subsets to reduce communication costs. The term of "Federated Learning" and its characteristics (i.e., training on mobile devices with unbalanced, non-IID data and wireless communication), was first introduced in [7]. Its baseline algorithm, FedSGD, involves each client taking a gradient descent step and communicating gradients to the server, which computes a weighted average of client gradients.

The communication overhead can be reduced through two basic concepts: 1) by reducing the communication frequency of weight updates (*communication delay*), i.e., multiple local iterations of weight updates are performed before transmission; or 2) by compressing the data to be transmitted, which typically involves one or several of the following approaches: (i) reduction of parameters, e.g., deleting elements (*pruning*), setting elements to zero (*sparsification*), or decomposing tensors, (ii) reducing the precision (i.e., *quantization*), or (iii) performing lossless compression such as entropy coding.

Communication delay: Federated Averaging (FedAvg) [7] is the most fundamental algorithm for reducing the communication frequency, in which the NN's gradients are not communicated after each forward-backward pass. FedAvg rather communicates updated weights as a generalized form of gradients after clients have performed multiple gradient descent iterations, and finally averages the weights on the server side. In [15], the number of local training iterations is adaptively adjusted so that the convergence of the learning system is optimized with respect to the wall-clock time. A communication-efficient delay method was proposed in [16], where parameters of the deeper located layers were communicated less frequently than those of the shallow layers.

Quantization: In order to reduce communication cost, TernGrad [17] quantizes the elements of the gradient vectors to three possible values $\in \{-1, 0, 1\}$, reducing the upstream communication by $32/\log_2(3) = 20.18\times$. However, at the cost of severe accuracy degradation in more complex tasks. The signSGD [18] method uses only the sign of the gradient elements as a biased approximation, resulting in $32\times$ compression. QSGD [19] reduces the communication cost of data-parallel stochastic gradient descent (SGD) in a multi-GPU environment. The authors propose a stochastic quantization scheme lowering the gradient's precision from 32-bit to 8-bit and subsequently apply *Elias* coding to the quantized gradients while fully preserving the accuracy of the trained networks. Although stochastic quantization is a convenient strategy, Suresh et al. [20] argue that the resulting error is sensitive to the gradient vector's distribution. To address this issue, a number of works proposes to randomly rotate the gradient vectors prior to quantization [20], [21], [22]. The rotated vector $x \in \mathbb{R}^d$ then tends to a normal distribution $\mathcal{N}(0, \frac{\|x\|_2^2}{d})$. Rotations (and inverse rotations on the server side) can be efficiently implemented by Walsh-Hadamard matrix products.

Sparsification methods select only a subset of the original neural update elements and set the remaining elements to zero, resulting in a sparse vector. Ström [23] presents an approach where only gradients with a magnitude greater than a certain threshold are communicated to the server. Since an appropriate threshold is hard to determine and model-dependent, top-k sparsification methods define a sparsity rate k, i.e., with $k = 0.1$ only 10% of the largest values are communicated [24]. On the contrary, Stich et al. [25] randomly select the communicated gradient elements (rand-k). Compared to quantization, sparsification techniques reduce the communication overload more aggressively. For instance, if proper compression error accumulation is used, the upstream communication can be sparsified to more than 99.9% in a number of use cases [26], [27]. However, considering also downstream compression using sparsification techniques is not as trivial: data heterogeneity on the client devices results in different sparsity patterns which can result in a dense matrix when averaged with other client updates on the server side.

Hybrid methods: Sparse Ternary Compression (STC) [27] applies sparsification, ternarization and *Golomb* encoding to weight updates. Due to extreme sparsification rates (>99.9%), compression errors are accumulated throughout the training process. Recently, Structured Sparse Ternary Compression (SSTC) [28] has been proposed as an extension of STC. To our knowledge, this is the only work that studies structured sparsity (here, in the granularity of convolutional filters) in the context of FL. However, the results are not very conclusive, since they are based only on experiments with tiny three-layer convolutional NNs (CNNs) and a more trivial MNIST [29] task. FedZip [30] is another approach which uses a pipeline of non-uniform quantization with k-means clustering, top-k sparsification and *Huffman* encoding.

Minors, namely Low-rank decomposition [31] and Federated Distillation [8], [32], [33], are described in the Supplemental Material B.

Surveyed state-of-the-art: There are a few very recent survey papers on communication efficiency in FL [34], [35], [36], providing an overview on communication challenges and state-of-the-art techniques for communication-efficient FL systems. The meta-studies show that there are several shortcomings in

the methodology of generating and presenting research results in the field. Many works use unrealistic learning scenarios, e.g., they do not provide benchmarks on tasks beyond the difficulty of MNIST [29] or, at best, CIFAR-10 [37], both of which are comparatively easy tasks to solve and less expressive in making a statement about the performance of a compression method. Alternatively, the use of non-standard datasets or a limited number of (possibly user-defined) models restricts comparability. Moreover, the computational cost of (de)compression is often ignored, even though it can be larger than the savings due to reduced communication [36]. The surveys also show that most of the methods are applied to gradient data rather than the weights of the models, although communication-efficient paradigms such as FedAvg rely on updates to the weights. Furthermore, in a majority of the approaches only the upstream communication is compressed, i.e., the communication from server to clients remains uncompressed, leaving a large potential for further savings in data communication unexploited.

Our contributions: Motivated by the previously mentioned shortcomings, we evaluate dNNC under a wide range of real-world models, datasets, and system settings including federated and split learning, e.g., as recommended by FedML [38], an open research library and benchmark for fair performance comparisons. Our training scenarios explore modern and standard NN architectures such as Vision Transformers [39], MobileNetV2 [40] or EfficientNets [41], which are ignored by the vast majority of works that continue to use VGGs [42] from 2014. The applied coding tools yielded encouraging results for compressing multiple data types, coming from different underlying distributions and structures. Specifically, we explore compressing weight updates but also gradients, and feature maps, which we evaluate in a split learning (SL) scenario. To our knowledge, no other published work has tackled the compression of both up- and downstream in SL communication. Other investigated learning scenarios include partial client participation, training from scratch and transfer learning, including large-scale image classification tasks such as ImageNet [43] or Pascal VOC [44]. Additionally, we compress both, up- and downstream communication in all experiments.

Unlike the related works, our preprocessing tools are tailored to the entropy coder, which exploits the properties of the preprocessed weight updates and effectively reduces the bitstream size. In summary, we propose a novel coding method for NN difference or update data, dNNC, which consists of highly optimized coding tools, including:

- A parameter update tree (PUT) mechanism, i.e., a high-level syntax that references a current NN coding unit to any previously coded unit
- New and updated data reduction tools, optimized to difference NN statistics, as well as (structured) sparsification methods that are leveraged in the coding stage, e.g., signaling skipped rows rather than encoding them
- A BatchNorm folding method for enhanced compression of BatchNorm updates in FL settings (FedBNF) that balances local and global statistics, preventing client drift
- A novel temporal context adaptation (TCA), specifically for improved context-adaptive binary arithmetic coding in incremental update scenarios.

Most of the dNNC tools have been included into the upcoming 2nd edition of the international NNC standard, which targets efficient compression of NN updates, while communication-specific tools like FedBNF ensure proper NNC operation in a broader range of FL applications. Finally, we investigate the energy consumption required for encoding and decoding and compare it with energy savings resulting from much shorter up- and download times of the compressed neural update data.

III. Coding and Communication Architecture

As presented in Section I, distributed NN systems with frequent update requirements have developed and are widely used. One major type of systems are federated learning or training scenarios, where a server distributes an initial "base" NN to a number of clients. These clients further train the NN on local data, resulting in updated local versions of the NN. Then, the local NN versions are sent back to the server, where all versions are aggregated (e.g., through parameter-wise averaging) into a new server-side NN version. Finally, this new version is again distributed to all clients and the training process is further iterated.

A. System Overview

An efficient way for reducing traffic in distributed NN systems is shown in Fig. 1, where difference NNs are generated and coded using the proposed dNNC method. First, an initial or base NN may either be coded and sent (as shown in Fig. 1, top, "Round 1"), or already be available at all clients. In the next step, difference neural networks (dNNs) are generated after the first local training round at each client. For this, the updated NN is subtracted from its previous version parameter-wise. dNNs can either be entire difference NNs or only contain a difference subset, such as last-layer updates, which is a common use case for transfer learning. Next, the dNNs are encoded with the proposed dNNC technology into a bitstream to be transmitted to the server. Then, the received client bitstreams are decoded, dNNs are reconstructed and aggregated to a new server-side dNN (cp. dNN_2 in Fig. 1). Subsequently, in communication round 2 (cp. Fig. 1), the server-side dNN is encoded and sent to the clients, where it is decoded and reconstructed towards full NNs, using our referencing mechanism, as explained in the next sub-section.

B. Parameter Update Tree (PUT)

The PUT is a novel mechanism that builds atop the recently published NNC standard ISO/IEC 15938-17 edition 1 [1]. For broad applicability in various distributed NN systems, the PUT provides a referencing mechanism that relates dNNs to previously sent versions to enable proper reconstruction of NNs on the server or clients. For this, the PUT mechanism utilizes the *high-level syntax* (HLS) structure of the NNC standard, where

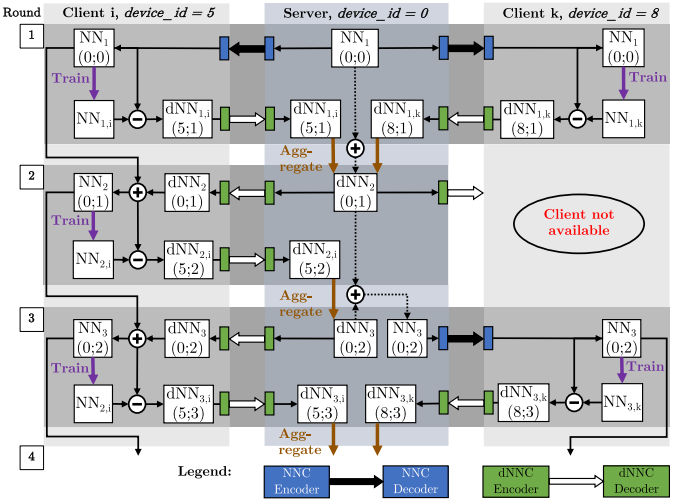

Fig. 1. Distributed dNNC setup and communication example, using Parameter Update Tree: Each coded and transmitted dNN is identified by the value pair (*parent_device_id*; *put_node_depth*). Left: active client; right: client failure.

the coded bitstream consists of different data units; in particular a compressed tensor resides within the payload of an *NNR*[1] *compressed data unit* (NNR_NDU) [45]. While the normative bitstream only contains the reference data, server and client devices can maintain their (non-normative) parameter update trees to allow for flexible communication and to address the different features and use cases in distributed NN systems:

- Federated learning, distillation, split and transfer learning, and neural network updates,
- complete or partial NN update, i.e., any layer or tensor-based update, such as last-layer updates, and
- synchronous and asynchronous NN communication.

The PUT mechanism represents incremental updates as a virtual tree structure. Given a particular parameter and its PUT, the parameter's base model values are associated to the root node of the PUT. An incremental update of a parameter corresponds to a child node attached to the root node. Any node of the PUT may further be updated by attaching child nodes in the same manner. For the coded transmission, a compressed dNN bitstream is associated with specific PUT syntax elements, namely *parameter_id*, *parent_device_id* and *put_node_depth*. The *parameter_id* identifies the respective parameter tensor within an NN, to which the dNN refers to. The *parent_device_id* specifies the server or client devices, and the *put_node_depth* refers to its distance to the root node in its virtual tree structure.

An example with two clients and a server is given in Fig. 1. Here, the server's *parent_device_id* is assumed to be 0, while the two clients are associated with *parent_device_id* equal to 5 and 8, respectively. For each communicated data unit or dNN, the value pair (*parent_device_id*; *put_node_depth*) is given. The PUT mechanism is shown for three training rounds for an active client on the left side and a partially inactive client on the right side. Whenever a server or client receives a dNN, it can be uniquely associated to its virtual PUT. In the example in Fig. 1, it is assumed that the root node is NN_1 with value pair (0;0). Consider round 2 at the left client i: When client i receives dNN_2 with (0;1), it can reconstruct NN_2, as the PUT parameter pair (0;1) indicates that a dNN is received from the server with a PUT node depth of 1, i.e., dNN_2 has to be added to NN_1. In turn, the server receives in round 1 two dNNs with value pairs (5;1) and (8;1), respectively. Therefore, the server identifies the *parent_device_id*s 5 and 8 as clients i and k and averages them, as they all have *put_node_depth* $= 1$ and thus belong to the same communication round.

On the right side in Fig. 1, a partially inactive client is assumed, i.e., after one training round the client becomes unavailable. The server averages the received updates from active clients, i.e., client i only in this example. In the following round 3, client k becomes active again, e.g., through notifying the server. As the client is missing data from the previous round, a full NN_3 is sent to client k. This can be achieved by the server through its virtual PUT, i.e., by identifying all required dNNs and add them to the PUT root node (cp. dotted line in Fig. 1). Client k then receives NN_3 and can resume operation by further train and create its new differential update $dNN_{3,k}$. Here, the *mps_parent_signalling_enabled_flag* syntax element, as specified in the NNC standard, is used to communicate that the bitstream represents a full NN and should not be added but replace.

Note, that the PUT mechanism can be applied to different synchronization mechanisms, e.g., also frequent full NNs might be sent to all clients to allow for synchronization points.

IV. Neural Network Coding Approach

Although our method focuses primarily on differential NN coding, a complete distributed learning system also requires a coding method for base or full NNs. We demonstrated this by introducing the PUT in the previous section, where full NNs are coded to provide the initial model to clients, to allow independent synchronization points at regular intervals, or for newly joining or rejoining clients. For full NN coding, the first edition of the NNC standard can be utilized (cp. Fig. 1, legend at the bottom). NNC edition 1 was released in August 2022 [1]. It established a toolkit of compression techniques for full neural networks and specifies the decoding process and the compressed representations of these networks. A complete overview of the different coding pipelines, features and tools of NNC edition 1 can be found in [45].

For coding differential neural networks (dNNs), our proposed dNNC coding method with encoder and decoder structure is shown in Fig. 2. The encoding process starts with parameter reduction methods, applied to the dNNs. These include unstructured and structured sparsification and pruning, as described in Section IV-A. When coding dNNs—in contrast to coding NNs — pruning and sparsification methods provide significant gains due to the different statistics of NNs and dNNs: Full or base NNs contain pre-trained parameters where it has been shown that the highest possible coding performance can be achieved by applying uniform quantization followed by DeepCABAC [46], the

[1]NNR is an obsolete abbreviation for the NNC standard edition 1, but is still used as a term in numerous syntax elements and stands for Neural Network Representation.

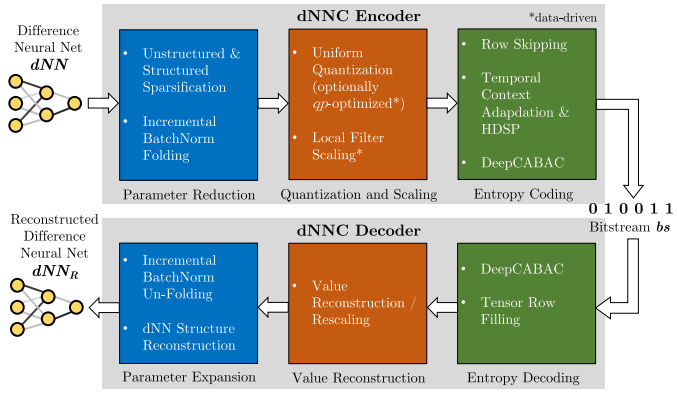

Fig. 2. Overview of the dNNC architecture. The encoder comprises three stages: 1) parameter reduction using sparsification and BatchNorm tools (FedBNF); 2) quantization and scaling via specific uniform quantization and local filter scaling (FS); and 3), entropy coding using row skipping (RS) and temporal tools like Temporal Context Adaptation (TCA) and History Dependent Significance Probability (HDSP), extensions of DeepCABAC. The decoder reverses these stages: 1) entropy decoding with DeepCABAC and tensor row filling; 2) value reconstruction; and 3) parameter expansion through BatchNorm un-folding and structure reconstruction.

entropy coder of NNC. In contrast, dNNs contain parameter differences and thus values with much smaller magnitudes. As also observed by [47], the distribution of dNNs is more centralized compared to NNs. In particular, many values can be set to 0, not only through quantization, but also through parameter reduction methods in order to increase coding gains.

Next in the encoding pipeline is a special BatchNorm folding method, FedBNF, (see Section IV-B). Then, quantization is applied, using specific adaptive quantization parameter (qp) optimization. Finally, the entropy coding stage applies tensor row skipping (RS) and temporal context adaptation (TCA) (described in Sections IV-C and IV-F, respectively) for optimized arithmetic coding through DeepCABAC. At the decoder (Fig. 2, bottom), the processing is inverted, starting with DeepCABAC decoding and tensor row filling. Then, the decoded weight updates are reconstructed and rescaled in accordance with FedBNF.

A. Sparsification and Pruning

In the following, we define the incremental NN weight updates as $\Delta W_\rho^{(t)} = W_\rho^{(t)} - W_\rho^{(t-1)}$, where W_ρ represents all model parameters, indexed by ρ, that are to be transmitted, and t denotes the index of the current communication round. Our proposed sparsification and pruning pipeline includes the following steps: 1) Parameter-wise statistics-adaptive sparsification (unstructured); 2) Filter-wise / neuron-wise sparsification (structured); 3) Pruning of sparsified structures (e.g., channel, neuron or whole layers updates) by row skipping (IV-C).

1) Unstructured Parameter-Wise Sparsification: In a first step, all absolute weight update values $|\Delta W^{(t)}|$ below the threshold θ_{qp} are set to zero, where

$$\theta_{qp} = \frac{4 + (qp \bmod 4)}{2} \cdot 2^{\lfloor \frac{qp}{4} \rfloor - 2}. \tag{1}$$

Here, the quantization parameter qp is an integer representation used in the NNC standard to specify equidistant quantization levels. Since quantization introduces a certain sparsity — subject to the used qp value — by assigning all absolute values $|\Delta W^{(t)}| < \theta_{qp}$ to the quantization level zero, θ_{qp} ensures that the sparsification method will contribute more to parameter sparsity than the follow-up quantization process.

Next, weight updates $\Delta W_\rho^{(t)}$ are sparsified further, depending on a threshold which is based on the mean values μ and standard deviations σ per parameter ρ:

$$\forall_\rho \quad \theta_\rho = \max(|\mu_\rho - \delta \cdot \sigma_\rho|,\ |\mu_\rho + \delta \cdot \sigma_\rho|),\ \text{s.t. } \theta_\rho > \theta_{qp}. \tag{2}$$

δ is a hyperparameter to control the intensity of unstructured sparsity. δ can be tuned iteratively until the model's performance falls below a tolerable level, e.g., -0.25% Top-1 accuracy. Alternatively, δ can be set such that it introduces a specific target sparsity in $\Delta W_\rho^{(t)}$.

2) Structured Filter-Wise Sparsification: In contrast to unstructured sparsification, structured sparsification aims at sparsifying whole regular groups of neighboring weight update elements, rather than setting individual elements to zero. In filter-wise sparsification, this group of elements is defined as all elements that contribute to one particular output feature of an NN layer. Consider a convolutional layer of dimension $[C_o, C_{in}, K, K]$, where C_o indicates the number of output channels (i.e., *filters*), C_{in} the number of input channels, and K the kernel size. One output feature is created by a filter which in turn is constituted by $C_{in}K^2$ filter elements. Thus, in filter-wise sparsification, we set $\varepsilon C_o C_{in} K^2$ update elements of a convolutional layer indexed by ρ to zero, with ε being the sparsity rate. The criterium for a filter update $\Delta\mathcal{F}$ to be sparsified is based on its absolute arithmetic mean value:

$$\forall_{\rho,m} |\Delta\bar{\mathcal{F}}_{\rho,m}^{(t)}| = |\Delta\bar{W}_\rho^{(t)}[m]| = \frac{1}{C_{in}K^2} \sum_{i=(0,0,0)}^{C_{in}-1,K,K} |\Delta W_\rho^{(t)}[i]|,$$
$$m \in [0, 1, \ldots, C_o - 1]. \tag{3}$$

After sorting the mean values of all C_o filter updates of a layer ρ in ascending order, the top εC_o filters are set to zero. Similar to unstructured sparsification, the structured sparsification can be fine-tuned by tuning ε or be fixed to a specific rate.

By default, we sparsify all filter updates that have absolute mean values below $\frac{0.9}{C_o}\sum_{m=0}^{C_o-1} |\Delta\bar{\mathcal{F}}_{\rho,m}^{(t)}|$, which is compatible with many neural architectures. For fully connected layers, C_o neurons directly map C_{in} input features to C_o output features, thus sparsifying εC_o neurons sets $\varepsilon C_o C_{in}$ update elements to zero. One-dimensional parameter types, such as BatchNorm or bias parameters are excluded from sparsification.

B. FedBNF: Federated BatchNorm Folding

Batch Normalization (BN) [48] is a technique to normalize the input activations of an NN layer per data batch for more stable training. Especially in modern and lightweight architectures such as the MobileNet family, BatchNorm parameters are ubiquitous. To recap, a BatchNorm module consists of a set of four vector parameters of length C_o: running mean E, running variance Var, and two learnable scale- and shift- vectors γ and β. The output y of an input x of a BatchNorm layer is defined

as

$$y = \frac{x - E(x)}{\sqrt{\text{Var}(x) + \epsilon}} \cdot \gamma + \beta \tag{4}$$

with ϵ being a constant factor, e.g., $1e-5$, for numerical stability. The role of BN in federated learning has hardly been explored. To mitigate feature shifts in non-IID data, FedBN [49] proposes to not synchronize local BN parameters with the global server model. SiloBN [50] synchronizes only the scale and shift parameters. Other work proposes to even replace BN with Group Normalization [51] in FL applications.

BatchNorm folding (BNF) [52] is a technique which defers BatchNorm parameters by merging them with the preceding NN weight layer. In the context of incremental update compression, BNF is not fully applicable because the original set of parameters is not reconstructable after folding and hence cannot be re-used to continue their learning process, e.g., at a certain client after receiving an aggregated and folded update of the BatchNorm modules by the server.

To implement BNF in a federated scenario (FedBNF), we propose to, first, BN-fold a local copy of the clients' model parameters by updating all γ and β parameters with their folded versions according to

$$\overset{*}{\gamma} = \frac{\gamma}{\sqrt{\text{Var}(x) + \epsilon}}, \qquad \overset{*}{\beta} = \beta - \overset{*}{\gamma} \cdot E(x). \tag{5}$$

The resulting representation of the folded network $\overset{*}{W}^{(t-1)}$ is stored locally. Next, the unfolded model $W^{(t-1)}$ is trained to generate $W^{(t)}$, then BNF is applied using (5), i.e., $\overset{*}{W}^{(t)} = \text{BNF}(W^{(t)})$, and finally $\Delta\overset{*}{W}^{(t)} = \overset{*}{W}^{(t)} - \overset{*}{W}^{(t-1)}$ is computed and transmitted to the server, which per se saves 50% of the BN partitions within the bitstream, since Var and E are skipped at encoding. Note that $\overset{*}{W}^{(t)}$ with $\text{Var} = 1$ and $E = 0$ generates identical inference results as $W^{(t)}$.

In the encoding stage, we use a dedicated data unit payload type, NNR_PT_BLOCK, for compressed data units (NNR_NDUs), which encodes the folded BN parameters together with their associated weight layers in one block, sharing one unit header. The header typically contains metadata such as layer names, device IDs, dimensions, and anything else required for reconstruction (the bitstream must be self-contained). Since, for instance, the dimension of the BN parameters can be derived from the dimensions of the associated weight parameters, the shared header can be represented more compactly compared to using separate headers.

On the server S, all received client updates $\Delta\overset{*}{W}_c^{(t)}$ with $c = i \in N$ are decoded, aggregated by averaging and added to its internal model state. Subsequently, the server update $\Delta\overset{*}{W}_S^{(t)}$, including the averaged statistics updates $\Delta\overset{*}{\gamma}_S{}^{(t)}$ and $\Delta\overset{*}{\beta}_S{}^{(t)}$, is broadcasted to the client instances, where client BN modules are updated as follows:

$$\gamma_c^{(t+1)} = (1-\eta) \cdot \gamma_c^{(t)} + \eta \cdot \sqrt{\text{Var}_c^{(t)} + \epsilon} \cdot \overset{*}{\gamma}_S{}^{(t)} \tag{6}$$

$$\beta_c^{(t+1)} = (1-\eta) \cdot \beta_c^{(t)} + \eta \cdot (E_c^{(t)} \cdot \overset{*}{\gamma}_S{}^{(t)} + \overset{*}{\beta}_S{}^{(t)}) \tag{7}$$

with $\overset{*}{\gamma}_S{}^{(t)} = \overset{*}{\gamma}_c{}^{(t-1)} + \Delta\overset{*}{\gamma}_S{}^{(t)}$ and $\overset{*}{\beta}_S{}^{(t)} = \overset{*}{\beta}_c{}^{(t-1)} + \Delta\overset{*}{\beta}_S{}^{(t)}$.

The momentum hyperparameter η ensures that local BN statistics adapted to client data domains are maintained, while preventing client drift by adding some level of aggregated global knowledge.

C. Tensor Row Skipping (RS)

Enforced by the quantization and the sparsification methods presented in Section IV-A, a large amount of update parameter values is set to zero. The resulting statistics are utilized in the entropy coding stage by skipping (i.e., pruning) matrix rows that are entirely zero. If a 2D matrix is arranged, such that each row corresponds to an output channel, i.e., a weight update of a specific filter $\Delta\mathcal{F}_\rho^{(t)}$ of an NN layer ρ, then all-zero rows can be omitted. For this, we developed a tensor row skip method, which introduces a specific skip_row_flag to be encoded for each row of the matrix. This flag determines, whether the values in a row are encoded (skip_row_flag $= 0$) or skipped (skip_row_flag $= 1$). Consequently, whenever the decoder receives a skip_row_flag equal to 1, no further binary symbols (bins) are decoded for the corresponding row, and all contained values are inferred to be zero.

D. Quantization Optimization (IQO)

Iterative quantization parameter (qp) optimization (IQO) is a straightforward search for an improved performance-bitrate trade-off. Per iteration, the qp value increases by qp_step. The qp value is an integer representation used in the NNC standard to specify a float step size s which is multiplied by each decoded integer value. s is derived from the integer quantization values qp and qp_density as follows:

$$\text{mul} = 2^{qp\_\text{density}} + (qp \bmod 2^{qp\_\text{density}}) \tag{8}$$

$$s = \text{mul} \cdot 2^{\lfloor \frac{qp}{2^{qp\_\text{density}}} \rfloor - qp\_\text{density}}. \tag{9}$$

We use qp_density$= 2$ such that $s = 2 \cdot \theta_{qp}$, see (1).

After increasing the qp value, the compressed $\Delta W^{(t)}$ is added to the prior base model $W^{(t-1)}$ to build an updated model $\hat{W}^{(t)}$. Then, the performance degradation of the updated model $\hat{W}^{(t)}$, potentially caused by quantization, is evaluated. If the obtained performance per remains above a reference performance per_{ref}, the qp value iteratively increases. As per_{ref}, we use the resulting test performance when $\Delta W^{(t)}$ is quantized using the initial qp value. Iterations stop if $per < per_{\text{ref}}$ for three consecutive iterations. Finally, the best performing qp among all iterations is used to continue training.

E. Filter Scaling (FS)

Local filter scaling equips convolutional (and fully-connected) layers with additional trainable scaling factors at each output element. The trained values of the scaling factors are then multiplied with the respective outputs of equipped convolutional filters and neurons. Fig. 3 illustrates the technology. As comprehensively studied in [53], our proposed filter scaling

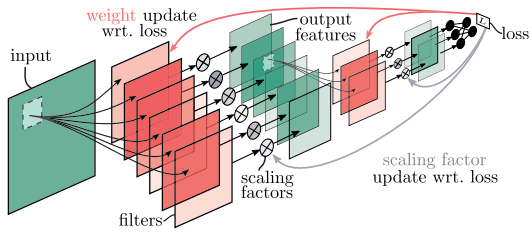

Fig. 3. *Filter Scaling* method: 1) A forward-backward pass with fixed scaling factors is performed, updating the model's weights. 2) The difference NN (dNN) with respect to the previous model state is calculated, compressed, and added to the previous model state. 3) The loss of the model — updated with the compressed dNN — is computed and backpropagated to update *only* the scaling factors. Step 3) may be repeated for n iterations to compensate for client domain shifts and the compression error in the dNN.

mechanism accelerates and regulates the entire federated learning process by enhancing some features in the filter space while diminishing others. Additionally, it motivates extra sparsity in the updates and thus achieves higher compression.

F. Arithmetic Coding Stage

As arithmetic coding core, DeepCABAC (context adaptive binary arithmetic coding) [46] is utilized. This method has been incorporated in NNC edition 1 [45], and is thus also used for base NN coding. DeepCABAC consist of three stages, which are described in more detail in [45], [46]:

1) *Binarization:* Each non-binary symbol or data element to be encoded (e.g., a quantized weight) is decomposed into a series of binary decisions (bins) in order to uniquely identify each symbol.
2) *Context modeling:* A probability model (*context model*) is assigned to each bin and associates a probability estimate with the bin.
3) *Binary arithmetic coding:* An arithmetic coding engine encodes each bin according to its estimated probability.

For binarization of a dNN, a parameter element to be transmitted is encoded in the following way: First, a bin called sig_flag is encoded which determines whether the element is equal to zero or not. If the sig_flag is equal to 1, a further bin sign_flag is encoded which denotes the sign of the element. Then a series of (usually ten) abs_level_greater_x flags signals whether the absolute value is greater than x. If this leaves a remainder, it is encoded using an exponential Golomb code.

Each context model employs a backward-adaptive state-based probability estimator, which maintains an internal state representing the probability estimate. After processing a bin, the state is updated based on the processed value: if a 1 is processed, the probability estimate for bin $= 1$ increases, while processing a 0 decreases the probability. The degree of this adjustment is controlled by adaptation rate parameters. DeepCABAC employs sophisticated context modeling, allowing it to adapt to a wide variety of input distributions. For encoding of the incremental parameter updates, dNNC adapts the context modeling stage of DeepCABAC in order to account for the temporal dependency of successive dNNs.

1) Temporal Context Adaptation (TCA): A proper choice of the context model can improve the coding efficiency if bins with similar statistics are assigned to the same context model. In dNNC, the temporal dependency of successive incremental updates can be exploited in the context modeling stage. For this, we developed the following context modeling scheme for the three CABAC elements: sig_flag, sign_flag and abs_level_greater_x flags. The context selection of a particular value q_i for each of the three elements is based on the corresponding or co-located value $q_{i,co}$ in the same layer of the previous incremental update. Here, one of two individual context models is assigned for each element, as follows:

- *For sig_flag:* If $|q_{i,co}| > 1$, context model $\mathcal{C}_{\text{sig},0}$ is chosen, otherwise $\mathcal{C}_{\text{sig},1}$.
- *For sign_flag:* If $q_{i,co} < 0$, context model $\mathcal{C}_{\text{sign},0}$ is chosen, otherwise $\mathcal{C}_{\text{sign},1}$.
- *For each abs_level_greater_x flag:* If $|q_{i,co}| \geq x$, context model $\mathcal{C}_{\text{ag}x,0}$ is chosen, otherwise $\mathcal{C}_{\text{ag}x,1}$.

Note that, if the previous incremental update is not available, e.g., if the current incremental update is the first to be transmitted or $q_{i,co}$ is equal to zero, the standard context modeling as described in [46] is applied.

2) History Dependent Significance Probability (HDSP): Parameter updates in successively sent dNNs show temporal correlations. For a particular parameter element, the probability of a significant (non-zero) update for this element depends on its updates from previous communication rounds. For this, HDSP adds a new context model for encoding the sig_flag of the dNN update elements. Compared to TCA, the selection is based on whether *any* of the updates of the parameter element were significant in previous communication rounds [54].

V. Experimental Results

This section presents our experimental results: First, Section V-A describes the experimental setup. Then, Section V-B presents in-depth dNNC coding results in different tool combinations. In particular, the results cover Federated Averaging (FedAvg) [7] and SplitFed [55], comparing training from scratch vs. transfer learning models, conducting ablation studies on sparsification and comparing vision transformers (ViTs) vs. CNNs in terms of communication efficiency. Our analysis includes a comparison of coding results with NNC edition 1 and other state-of-the-art approaches. Next, we take a more in-depth look at partial client participation, i.e., a more realistic setting where only a fraction of clients sends updates to the server per round. Afterwards, we show some advantages and limitations of data-driven optimization which aims to further reduce the communication overhead of the system by utilizing validation data. Finally, in Section V-B3, results on energy and runtime measurements are presented, alongside considerations justifying the additional computational effort introduced by our proposed dNNC pipeline, as well as exploring potential applications.

A. Experimental Setup, Test Data and Assessment

We perform experiments using CIFAR-10 and CIFAR-100 [37], ImageNet-200 (for which we randomly sampled 200

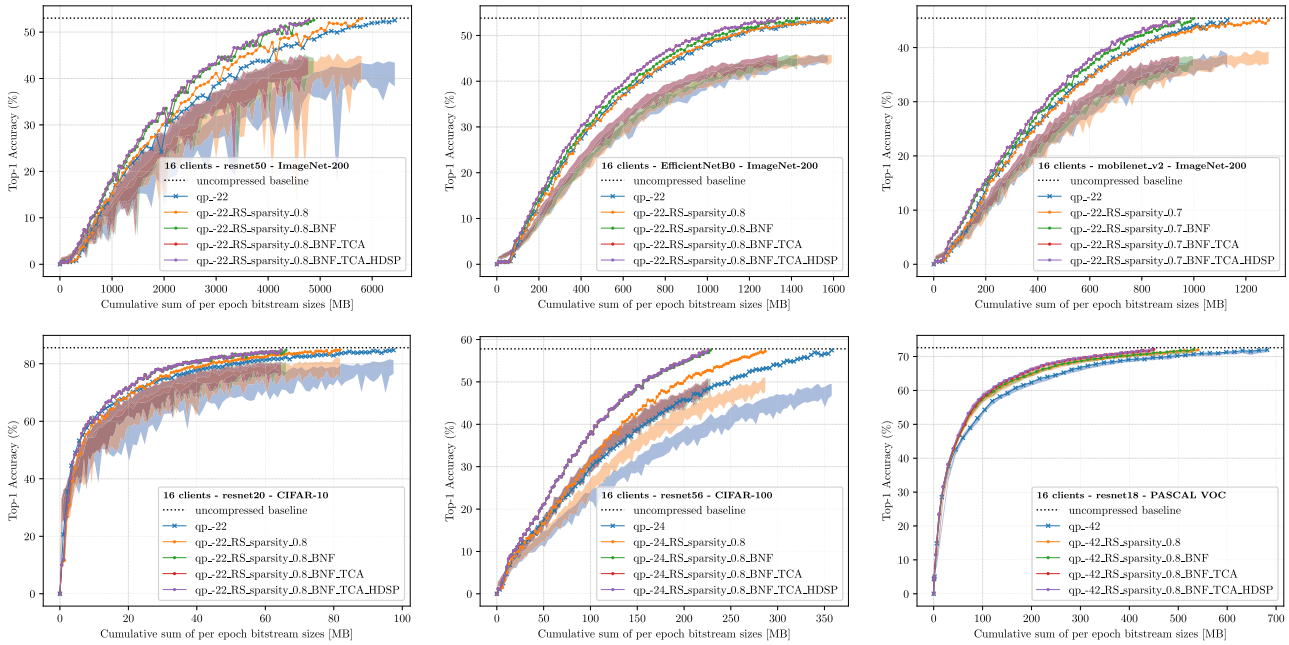

Fig. 4. Coding results with successively enabling data-free compression tools in the following order: quantization (qp) only $\rightarrow$ + sparsification + row skipping (RS) $\rightarrow$ + FedBNF $\rightarrow$ + TCA $\rightarrow$ + HDSP. In each experiment, 16 clients train a global model via the FedAvg paradigm. The transparent curves denote the variance of the local client performances per communication round and the line plots with markers, those of the global model.

classes from the ImageNet-1k [43] with 500 samples per class) and Pascal VOC [44] datasets. The batch size is 32 in all FL settings and 64 in the SL settings. An Adam optimizer [56] with an initial learning rate of $1e-3$ is used for training from scratch and $1e-5$ for transfer learning. Unless otherwise specified, unstructured sparsity was fixed to 80%. Structured sparsity was chosen such that all filter updates below the average filter mean $|\Delta\bar{\mathcal{F}}_{\rho}^{(t)}|$ were sparsified (and row-skipped). For FedBNF, we used a momentum η in a range [0.2, 0.4]. We use error accumulation in all FL use cases. For a more in depth description of the experimental environment, models, hyperparameters, the datasets employed, and data preprocessing, we refer to the Supplemental Material C.

B. Data-Efficient Distributed Learning Communication

1) Coding Results of Federated Averaging (Basic Setting): First, we discuss the coding results of the whole training process shown in Fig. 4. Here, each marker in the graphs corresponds to one communication round, indicating the Top-1 accuracy of the global server model after aggregating the clients' updates and the cumulative sum of communicated data. Here, the upper-leftmost curve in the charts yields the most communication-efficient FL with minimum cumulative data rate at non-degrading system accuracy.

The transparent curves display the variance of the local client performances on the test data set. Each color represents a particular compression pipeline. As a default setting (blue), we use uniform quantization with a step size of $s = 2 \cdot \theta_{qp}$ according to (1) (e.g., $qp = -22$ corresponds to ≈ 0.0234) and arithmetic coding as described in Section IV-F.

By successively switching on tools, we demonstrate the individual impact of each tool on the overall coding performance. In a first step, we apply 80% unstructured sparsity, our default structured sparsification and the row skipping (RS) mechanism to the weight updates (orange). As a second tool, we turn on our proposed federated BatchNorm folding (FedBNF) along with its corresponding coding block structure (green). This is followed third by TCA (Temporal Context Adaptation) (red) and fourth by HDSP (History Dependent Significance Probability) (purple).

Different tool settings show different effects: For instance, sparsifying 80% (70% in the MobileNetV2 use case) of the update elements can either reduce the overall number of bytes transmitted by more than 20%, as in the CIFAR-100 use case with ResNet-56, or might even increase the overall number of bytes, as in the MobileNetV2 use case with federated learning of ImageNet-200. In the latter case, the neural architecture appears to be more sensitive to structured sparsification methods of weight updates, requiring more communication rounds to achieve the baseline accuracy of the uncompressed scenario (shown as a dotted line in the diagrams). This could be due to the extensive use of depth- and group-wise convolutions in MobileNets, where information exchange between channels is limited, so that updates of entirely sparse channels lead to slower convergence of the system. On average, the sparsification methods with a fixed rate of 0.8 reduce the communication traffic by 11.2%. Our FedBNF method achieves substantial reduction in communicated data in all use cases: On average, 17.7% can be saved. In the learning-from-scratch use cases, coding gains by TCA and HDSP are comparably small, however their application is lossless. On average, TCA reduces the data traffic by 3.4% and HDSP by 0.02%.

Table I shows the number of bytes communicated and the communication rounds required to achieve at least 99% of the target performance ϕ, which is the peak performance of the uncompressed scenario. The "uncompressed communication"

TABLE I
OVERALL CODING RESULTS FOR 100 COMMUNICATION ROUNDS (50 IN TRANSFER LEARNED SCENARIOS)

model	paradigm	#clients	data	uncompressed communication			compressed communication			
				acc. [%]	$\#_{\text{rounds}}$	$\sum$data [GB]	Δacc. [%]	$\Delta\#_{\text{rounds}}$	$\sum$data [MB]	CR (*w/o HLS*) [%]
ResNet-20	FL	16	CIFAR-10	85.59	99	3.42	-0.84	+4	65	1.89 (*1.72*)
ResNet-56	FL	16	CIFAR-100	57.79	100	7.86	-0.30	+18	226	2.87 (*2.61*)
ResNet-50	FL	16	ImageNet-200	53.00	75	229.61	-0.42	+14	4,778	2.08 (*2.07*)
MobileNetV2	FL	16	ImageNet-200	45.44	97	30.79	-0.11	-3	944	3.07 (*3.01*)
EfficientNet-B0	FL	16	ImageNet-200	53.78	90	49.12	-0.04	+1	1,334	2.72 (*2.67*)
ResNet-18	FL^{τ}	16	Pascal VOC	72.61	49	70.16	-0.48	-3	446	0.64 (*0.63*)
ViT-B/16	FL^{τ}	16	Pascal VOC	78.54	20	219.68	-0.56	+26	246	0.11 (*0.11*)
ResNet-20	SL	20	CIFAR-10	82.00	99	655.36	-0.54	-9	18,049	2.75
ResNet-56	SL	20	CIFAR-100	48.14	99	2,621.44	-0.30	-2	88,570	3.38
MobileNetV2	SL	20	ImageNet-200	43.86	96	6,021.06	-0.17	-16	261,328	4.34
EfficientNet-B0	SL	20	ImageNet-200	52.28	69	11,239.31	-0.74	0	454,019	4.04

FL refers to federated learning from scratch, FL^{τ} to federated transfer learning and SL to the SplitFed communication scheme, "w/o HLS" shows the compression ratio (CR) without accounting for high-level syntax.

TABLE II
TRADE-OFF BETWEEN STRUCTURED AND UNSTRUCTURED SPARSIFICATION

	CIFAR-10 / -100		Pascal VOC
Sparsification method	ResNet-20	ResNet-56	ResNet-18
qp-induced	97.46 MB (t=85)	357.76 MB (t=98)	681.74 MB (t=46)
structured (80%)	—	—	-52.01% (t=50)
unstructured (80%)	+1.25% (t=91)	-8.11% (t=107)	-2.38% (t=45)
structured	-0.80% (t=86)	-6.09% (t=104)	-3.25% (t=45)
+ row skipping	-1.01% (t=86)	-7.87% (t=104)	-3.42% (t=45)
+ unstructured (80%)	-16.01% (t=105)	-19.85% (t=118)	-20.79% (t=46)

column shows the number of communication rounds ($\#_{\text{rounds}}$) at which ϕ was achieved (within the overall budget of rounds that is 100 for training from scratch and 50 in the transfer learning setting). The $\sum$data-column reports the sum of communicated neural update data within $\#_{\text{rounds}}$. In the "compressed communication" column, Δacc. indicates the loss in accuracy which is constrained to be less than 0.01ϕ. Additionally, $\Delta\#_{\text{rounds}}$ indicates the number of communication rounds differing from the uncompressed scenario. We chose the quantization parameter qp such that the compressed scenarios converge within 120 rounds and 60 rounds for the "from scratch" and "transfer learning" settings, respectively. The reported compression ratio (with $\text{CR} = 100 \times \frac{\text{compressed data}}{\text{uncompressed data}}\%$) includes high-level syntax (HLS), e.g., tensor name strings, identification numbers, coded data unit structures, etc., which can account for up to 9% of the transmitted data (e.g., in the CIFAR use cases). This is mostly caused by the tensor name strings, which could be replaced by integer identifiers if server and clients specified a lookup table for reconstruction.

2) Ablation Studies on dNN Sparsification: We examined the effects and coding contribution of unstructured sparsification, structured sparsification, and tensor row skipping individually in an ablation study (as only their combined effect is shown in Fig. 4's coding results). Accordingly, Fig. 5 shows example sparsification results for ResNet-56 on CIFAR-100 and ResNet-20 on CIFAR-10. The curves labeled "struct_mean" represent our default structured sparsification, which at communication round t sets all filter updates $\Delta\mathcal{F}_\rho$ of a layer ρ that have absolute mean values below $\frac{0.9}{C_o}\sum_{m=0}^{C_o-1}|\Delta\bar{\mathcal{F}}_{\rho,m}^{(t)}|$ to zero (cp. IV-A2). Curves labeled "RS_struct_mean" show the same configuration with row skipping (RS) enabled, resulting in a reduction of transmitted data of up to 7.8% compared to the sparsification-free setting (cp. Table II). For the ResNet-56 use case, enabling row skipping reduces the amount of communicated data by 1.8%. In comparison, curves labeled "struct_sparsity_0.80" show the results when 80% of each layer's filter updates are sparsified,

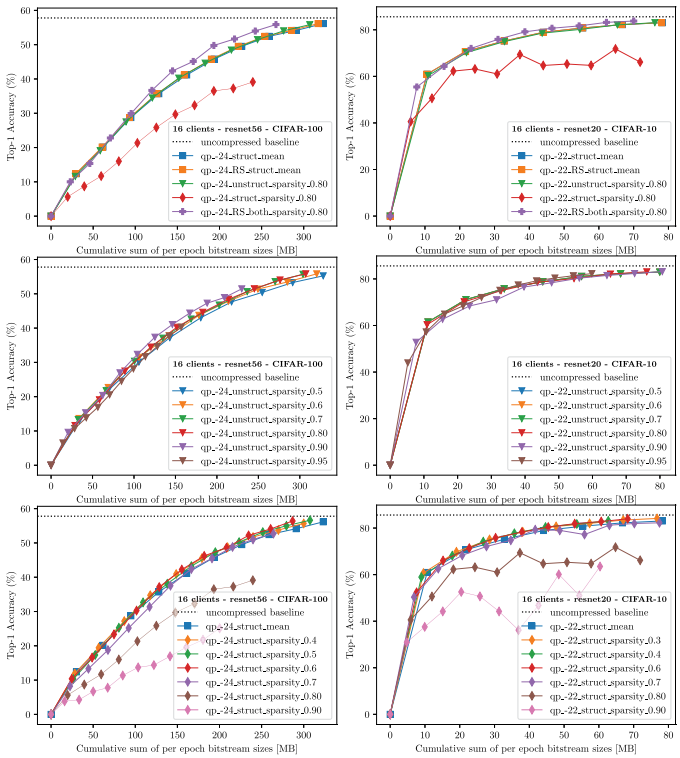

Fig. 5. Ablation study on the effects of unstructured and structured sparsification and tensor row skipping. Every 10th datapoint plotted (for visibility).

resulting in slower convergence (in fact, the train runs do not converge within the given budget of communication rounds). In contrast, setting 80% of the update elements to zero using our *un*structured sparsification method does not result in slowed convergence (cp. curves labeled "unstruct_sparsity_0.80"). Finally, the synergistic combination of structured sparsification, row skipping, and unstructured sparsification (cp. curves labeled "RS_both_sparsity_0.80") results in the most communication-efficient runs. For instance, it achieves up to 19.9% reduction in communication cost for ResNet-56, while maintaining the accuracy of the uncompressed baseline.

In the federated transfer learning setting, the behavior of sparsification tools varies significantly. Due to the relatively small learning rate and minimal changes in client weights, the model differences ΔW inherently exhibit high sparsity, particularly after uniform scalar quantization. Consequently,

unstructured sparsification alone has minimal impact, while structured sparsification can yield higher overall compression rates than a combination of both, as evident in Table II (please refer to Supplemental Material D-A for detailed result plots).

3) Coding Results of FedAvg (Transfer Learning Setting): In transfer learning, the client networks already have some knowledge in a related domain. Here, we install networks that are pre-trained on ImageNet-1k and then learned on the data domain of Pascal VOC. For this setting, we deploy ResNet-18 and ViT/B-16 as NN architectures. First, we compare the training pattern of the ResNet-18 setup with the use cases of the previous section, i.e., training from scratch. Then, we compare the training pattern of ViTs vs. ResNets.

Relative to learning-from-scratch, in transfer learning the individual tools in our proposed compression pipeline contribute quite differently to the overall compression rate: For example, FedBNF has a rather limited effect (1.5% reduction) which is also due to the reduced number on BatchNorm modules in ResNet-18. In contrast, TCA and HDSP have a much larger impact with 15.4% and 0.9%, respectively. Another salient point is that the local client accuracies are at the same level as the global server model and there is much less variance in client accuracies (cp. Fig. 4). With compression ratios of 0.64% and 0.11% (cp. Table I), the transfer learning scenarios are generally more compressible. This is consistent with the intuition that there is only some adjustment of certain weights to the new, unseen data, i.e., the elements of the absolute weight update have a smaller magnitude on average, which introduces high sparsity. Since the NN models converge faster in the transfer learning setting, gradient descent is — already in early update rounds — more directed and smoother, which is beneficial to Temporal Context Adaptation (TCA). It works best, when the elements to be encoded behave similarly in subsequent rounds of communication, e.g., if they always have the same sign and range. This increases the likelihood of the use of certain context models, which in turn allows the coder to represent the weight update elements with fewer bits.

ViTs [39] may set new standards in FL: compared to ResNets, Vision Transformers (ViTs) converge faster in our experiments, e.g., after only 20 rounds compared to 49 in the ResNet case, and achieve higher accuracies, e.g., 78.5% accuracy versus 72.6% (cp. Table I). Moreover, the neural update data to be transmitted is much more compressible: using the default compression pipeline (e.g., without FedBNF, since not directly applicable to ViTs), ViT communication can be reduced by 893×, while ResNet-18 communication is reduced by 103× (please refer to Supplemental Material, Fig. D.1, for result plots). To match the best global accuracy of ResNet-18, the ViT experiment requires 82% less data to be transmitted and 30% fewer rounds of communication to be performed. However, in terms of on-device complexity, the ViT architecture requires considerably more computational and memory resources. ViTs could give rise to new compression technologies due to their decomposable architecture, and we consider it a part of future studies to further investigate ViT dNN-compression in FL.

4) Comparison of Coding Results With NNC Edition 1 and State-of-The-Art: Compression techniques in FL lack comparability as discussed in the surveyed state-of-the-art and our contributions Section II. However, comparing approaches is essential; Sparse Ternary Compression (STC) [27] and FedZip [30] are often considered superior. E.g., STC and FedZip excel established benchmarks like FedAvg [7], signSGD [18], and DGC [26] in the trade-off between number of communication rounds and number of bits communicated. FedZip uses a pipeline of top-k sparsification, non-uniform k-means quantization and Huffman encoding. Actually, FedZip uses three encoding methods: Huffman code, encoding of address positions in address table and encoding differences of address positions in address table. For each round t we chose the smallest resulting bitstream size among these three options. In STC, top-k sparsification is basically followed by uniformly quantizing the remaining top-k elements to their mean population magnitudes such that top-$k \in \{-\mu_{\text{STC}}, 0, \mu_{\text{STC}}\}$ (i.e., ternarization), followed by Golomb code.

Fig. 6 displays coding results varying the sparsity for STC and FedZip, compared to dNNC's default setting (as described in Section V-B1). Complementary data on communicated bits per method is detailed in Table III. Notably, we benchmark dNNC without accounting for high-level syntax (HLS) in the bitstream, consistent with STC and FedZip. $\#_{\text{rounds}}$ indicates the communication rounds until convergence (or best accuracy within specified budget of rounds: $t = 120$ for FL from scratch and $t = 60$ for transfer FL).

Our dNNC method achieves convergence in all four use cases, requiring the least amount of data transferred. Except for CIFAR-100, where FedZip converges faster, dNNC exhibits the fastest convergence in the other use cases. For the implementation of the other coding methods we used publicly available software repositories of the STC and FedZip authors (we refer to the Supplemental Material C-E for details).

Using FedAvg [7] and FedAvg + NNC edition 1 [45] compression as baselines, Table IV demonstrates the superiority of our proposed preprocessing and additional coding tools. It shows the average data transmitted per communication round between 16 clients and a server (as executed in the previous Sections V-B1 and V-B3).

Our proposed dNNC compression pipeline reduces the FedAvg communication overhead by 98% on average. Compared to the NNC-compressed baseline, the data-free dNNC tools further reduce the amount of communicated data by 37%, on average. On top of this, data-enhanced approaches can reduce the average data transmitted per round even further, e.g., to 0.46 MB for the ResNet-20 use case with iterative qp optimization enabled, which then corresponds to a 60% saving in communication compared to the NNC edition 1 baseline (cp. the following Section V-B5 on fine-tuned NN compression).

5) Fine-Tuned Neural Update Compression Using Data: At the cost of additional local computation, the neural update data can be compressed further. We have implemented two of our approaches for data-driven optimization, namely filter scaling (FS) and iterative qp optimization (IQO). Obviously, it is also

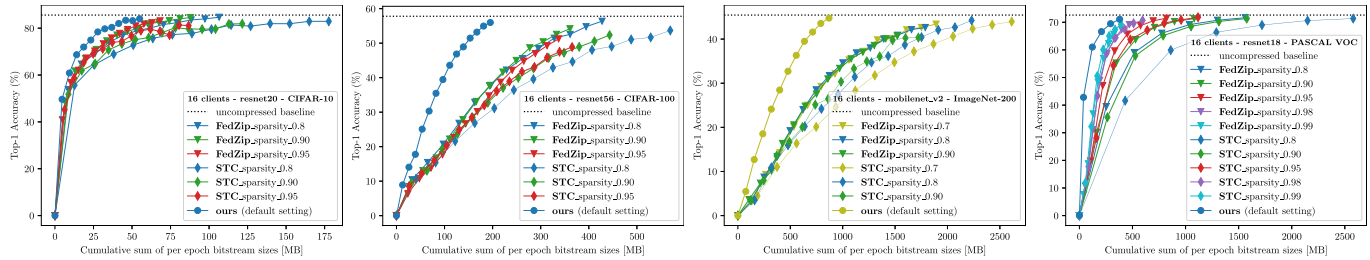

Fig. 6. Comparative results with state-of-the-art compression schemes for FL, Sparse Ternary Compression (STC) [27] and FedZip [30]. Each color represents a sparsity rate, the main tuning parameter for STC and FedZip. "Default setting" implies that all data-free dNNC tools are enabled, according to the configuration outlined in main coding results Section V-B1. Every 8th datapoint plotted (for visibility).

TABLE III
COMPARISON OF DNNC WITH FEDZIP AND STC (SPARSE TERNARY COMPRESSION)

	CIFAR-10, ResNet-20			CIFAR-100, ResNet-56			ImageNet-200, MobileNetV2			Pascal VOC, ResNet-18		
	Δacc. [%]	#rounds	∑data [MB]	Δacc. [%]	#rounds	∑data [MB]	Δacc. [%]	#rounds	∑data [MB]	Δacc. [%]	#rounds	∑data [MB]
STC [27]	-1.59 ×	116	184.06	-3.20 ×	119	604.94	-0.45 ✓	117	2,329.87	-0.48 ✓	57	1,137.53
FedZip [30]	-0.67 ✓	105	88.81	-0.34 ✓	111	456.72	-2.04 ×	112	1,891.23	-0.69 ✓	53	1,197.86
dNNC	-0.84 ✓	103	58.90	-0.30 ✓	118	205.64	-0.11 ✓	94	927.88	-0.48 ✓	46	443.29

✓means the training converged within the given number of communication rounds t, × means the opposite

TABLE IV
AVERAGE COMMUNICATED DATA IN MEGABYTES PER COMMUNICATION ROUND WITH 16 CLIENTS

	CIFAR-10 / -100		ImageNet-200			Pascal VOC
	ResNet-20	ResNet-56	ResNet-50	MobileNetV2	EfficientNet-B0	ResNet-18
FedAvg [7]	34.70	79.80	3068.28	321.82	551.14	1433.14
+ NNC† [45]	1.16	3.65	101.32	12.69	20.93	15.28
+ **dNNC**	0.63	1.91	55.16	10.03	14.84	9.94

†Neural Network Coding standard ISO/IEC 15938-17 edition 1 applied to difference updates as communicated in FedAvg.

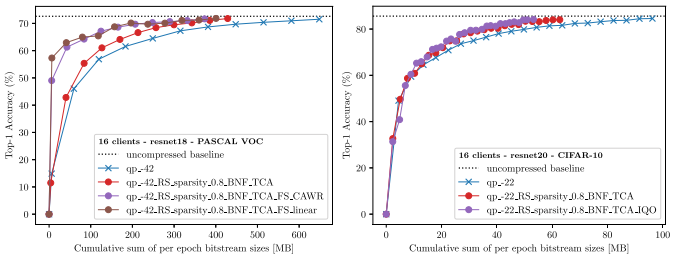

Fig. 7. Fine-tuned, i.e., data-enhanced, neural update compression. **Left:** Filter Scaling (FS); **right:** Iterative *qp* optimization (IQO). Every 4th datapoint plotted (for visibility).

possible to fine-tune additional hyperparameters, such as the sparsification rate or the BNF momentum η, so that it changes adaptively during training. However, these experiments can be quite time-consuming and are rather straightforward from an algorithmic background.

Fig. 7 presents the resulting training curves of FS and IQO. For the proposed filter scaling method, we use an initial learning rate of $1e-2$ for training the scaling parameters. The scaling parameters are trained separately for additional 5 local rounds while the rest of the NN parameters is fixed. During training, the learning rate for scaling parameters is scheduled by i) a cosine annealing learning rate scheduler with warm restarts (CAWR) or ii) by a linear scheduler. CAWR achieves higher accuracies in the later communication rounds whereas the linear scheduler is more beneficial in the early rounds. Both settings drastically speed up training in the sense that already in early communication rounds comparably high global accuracies are achieved (e.g., >60% acc. in round 5, cp. Fig. 7, top). The target accuracy of the ResNet-18 transfer FL use case is exceeded in round 43, with a total of 405 MB of data transferred, a reduction of 9% compared to the best data-free setting (cp. Table I).

For IQO, only model inference is required, no additional training. However, it may take several iterations of increments of the qp value to find an optimal solution in terms of rate-distortion. We use $qp\_step = 1$ as the increment and tolerate a model degradation of up to -0.25%. Compared to the best data-free setting, enabling IQO in the ResNet-20 FL use case further reduces the amount of data to be communicated from 65 MB to 54 MB (-17%).

6) Partial Client Participation / Asynchronous Communication: In distributed learning scenarios, often only a fraction of the clients is able to simultaneously transmit their dNNs to the server. Whether for connectivity or other capacity restrictions, it is crucial that the system can converge under such conditions. Therefore, we study the impact of the proportion of participating clients. Intuitively, a gradual reduction in active clients should reduce the total amount of data transmitted, but interestingly, we observe the contrary when the participation p is less than 25% (using 64 clients). We found that the server update is less sparse in that case. This is because only a portion of the dataset is effectively trained per round. Thus, for an unrepresentative subset of clients, certain weight update elements remain that interfere with minimizing the global loss function (and that might be averaged out with higher client participation p). Overall, the best performing experiments in terms of data-efficient and fast learning are those with coarse quantization and medium client participation (e.g., $qp = -24, 0.25 \leq p \leq 0.5$). In summary, it is a trade-off between the number of active clients and the degree of compression. A low number of active clients and high compression concurrently are associated with additional rounds, slower convergence, and potentially more data traffic. For details on the

implementation and result plots we refer to the Supplemental Material D-C.

7) Coding Results of SplitFed Learning Settings: The SplitFed (SFL) [55] approach combines the benefits of FL [7] and Split Learning (SL) [9] by parallelizing SL communication through enabling clients to engage with i) a main server that hosts the respective server-side portions of the client networks and ii) a fed server which aggregates the client-side model portions in a FedAvg manner. The communication then involves sending activations, called *smashed data*, of the so-called *cut layer* to the server, and subsequently receiving the gradients of the smashed data from the server which hosts the remaining parts of the client models.

With this different communication scheme, new challenges for NN data compression arise. First, the amount of communicated data can be much higher compared to FL, because the activation space is larger than the latent weight space. Furthermore, it scales with the size and number of input samples (e.g., smashed data from convolutional layers sums up to [batch_size $\times$ output_channels $\times$ feature_height $\times$ feature_width] $\times 32$ byte (for full precision), which can be in the range of several dozen of megabytes for one single forward-backward pass of one client). Second, local learning (for communication delay) is not possible because in the default S(F)L setting, client splits do not include the classifier part of the net, and thus any non-transmitted smashed data directly leads to the exclusion of the associated mini-batch from the overall training. Also, due to the nature of the underlying data, different distributions apply; for example, feature maps in the upstream domain may have much larger values and variances compared to gradient data in the downstream domain, depending on, e.g., the activation functions and learning rates used.

In our experiments, we split the ResNets and EfficientNet after their first block and the MobileNetV2 architecture after the second block, so that the resulting client portions are between 10 kB and 68 kB in size, which fits well in the RAM of typical IoT device processors such as microcontrollers. Since smashed data and gradients have quite different distributions, and for speed-up reasons, we constrain the number of quantization levels dependent on the underlying data's dynamic range. We tested several bitwidths $b \in 2, 3, 4, 6$ bit (cp. Supplemental Material D.3). As a clipping range, we used the 99.95th percentile $P_{99.95\%}$ of the absolute values of the underlying data distribution. The step size is $s = P_{99.95\%}/2^{b-1}$ for unsigned representations (e.g., positive valued ReLU activations) or $s = P_{99.95\%}/(2^{b-1}-1)$ for signed representations. From these step sizes we derive a $qp = \lfloor s \cdot 2^{\lfloor \log_2 s \rfloor} + 4(\lfloor \log_2 s \rfloor - 1) \rceil$ with $\lfloor\ \rceil$ being the rounding function. It turned out that quantization did not decrease the global accuracy when using $b = 3$ bit and $b = 4$ bit for the CIFAR and ImageNet experiments, respectively. For the upstream, we obtained an average $qp \approx \lfloor 0.51 \rceil$ within a range of $[-3, 5]$. For the downstream, we obtained an average $qp \approx \lfloor -56.87 \rceil$ within a range of $[-72, -27]$. Although much finer quantization levels are applied to the gradient data, they are on average more compressible than the smashed data, e.g., up to $2.6\times$ in the case of ResNet-20, which might be different if the cut layer was at a deeper location in the network and the feature maps might be sparser. However, this would also increase the computing and memory requirements of the client devices. Table I (bottom rows) shows the compression results for training 20 clients in a SplitFed fashion. Interestingly, the compressed communication leads to faster convergence (overall less communication rounds required).

TABLE V
ENERGY CONSUMPTION FOR FL COMMUNICATION AND TRAINING

	VOC, ResNet-18		CIFAR-100, ResNet-56			
	$N=4$, $t=60$		$N=4$, $t=120$		$N=16$, $t=120$	
process	c [kWh]	uc [kWh]	c [kWh]	uc [kWh]	c [kWh]	uc [kWh]
download	0.11	8.95	0.03	1.00	0.10	3.99
decode	0.39	0.00	0.22	0.00	0.94	0.00
train/agg	38.14		70.92		109.94	
encode	0.68	0.00	0.50	0.00	1.70	0.00
upload	0.19	16.51	0.05	1.84	0.16	6.25

c...compressed, uc...uncompressed.

C. Energy Efficiency and Runtime Analysis of dNNC

dNNC not only reduces communication bandwidth, but even provides net energy savings. For the experiments, we used a Jetson Xavier NX series device for energy consumption monitoring and additionally estimated the energy required for communication between server and clients by summing the energy consumed by the routers and the client instances (i.e., the Jetson Xavier NX) during the download and upload of weight updates. We follow the recommendations by Qiu et al. [57] and formulated the overall energy consumption for communication E_{com} as

$$E_{\text{com}} = \sum_{j=1}^{t} \sum_{i=1}^{N} \mathbb{I}_{c_{i,j}} \cdot \left(\frac{bs_D}{r_D} + \frac{bs_U}{r_U} \right) \cdot (E_R + E_{\text{idle},i}), \quad (10)$$

with t being the total number of communication rounds, N the number of clients, $\mathbb{I}_{c_{i,j}}$ an indicator function for client i being active in round j, bs_D and bs_U the down- and upstream bitstream sizes, r_D and r_U the download and upload speeds, E_R the power of the router and E_{idle} the power of the hardware of the idle clients. In our setting, we measured an E_{idle} of 2.25 W for the Jetson Xavier NX and calculated an $E_R = 10.2$ W, $D = 66.52$ MBit/s and $U = 22.51$ MBit/s, in accordance with [57] (i.e., the median country-specific download and upload speeds as reported on *Speedtest*[2] and router power as reported on *The Power Consumption Database*[3]). The energy consumption for encoding E_{enc}, decoding E_{dec} and training E_{train} is directly measured on the Jetson board as described in the Supplemental Material C-F.

Table V shows the estimated and measured energy consumption for federated learning the VOC and CIFAR-100 use cases. Overall, the energy savings for transmitting dNNs at a fraction of their original size out weight the additional energy required for en- and decoding, i.e., $(E_{\text{com},c} + E_{\text{enc}} + E_{\text{dec}}) << E_{\text{com},uc}$. Thus, dNNC provides significant savings in the overall E_{com}, especially for the larger ResNet-18 where E_{com} is in the regime of E_{train}, dNNC saves up to 94% of the required kWh for communication. Fig. 8 displays the energy consumption separately for

[2]https://www.speedtest.net/global-index
[3]http://www.tpcdb.com/list.php?page=1&type=11

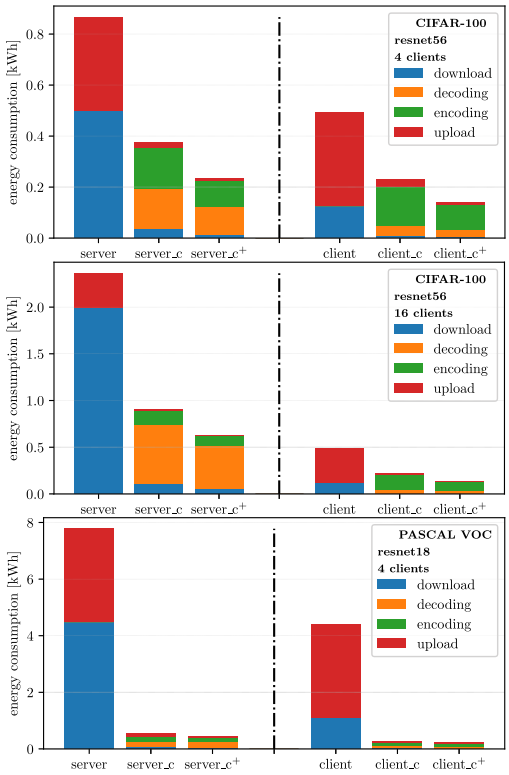

Fig. 8. Energy consumption on server side and average energy consumption of clients for downloading, decoding, encoding and uploading neural network updates. The energy consumption is aggregated over 60 communication rounds for the VOC task and 120 rounds for the CIFAR-100 task. Labels with suffix "_c" mark experiments with default compression, i.e., using uniform quantization and the DeepCABAC codec. Experiments labeled with suffix "_c$^+$" show energy consumption using our proposed dNNC compression pipeline with data-free tools, i.e., sparsification, row skipping, FedBNF and Temporal Context Adaptation.

server and client side, distinguishing between energy consumption for down- and upload as well as encoding and decoding of NN updates. Note that, at the server, encoding is only required once, while decoding is required for N received client updates.

Runtime analysis is used in ISO/IEC MPEG standardization activities as a means for comparing and assessing encoder and decoder complexities, given that comparable experiments were carried out on similar hardware and resource utilization settings, e.g., parallelization. Similar to this approach, we also express the coder's time complexity as a function of the input data length n, i.e., the coding algorithm has a time complexity of $\mathcal{O}(n)$, i.e., it processes each symbol of the input sequence once. Note, that this is a simplification and the actual performance may be affected by specific coding choices or implementation details. Table VI shows the measured required average runtimes on the Jetson board which is indicated as a real-world hardware platform for FL applications by the FedML [38] open research library. Runtimes are averaged over all clients $i \in N$ and communication rounds $j \in t$ and complement the energy consumption in Table V. Additionally, Table VII presents runtimes for learning under the environmental settings of our main experiments in Section V-B1.

TABLE VI
AVERAGE PROCESSING TIMES FOR DECODING, TRAINING AND ENCODING PER CLIENT DEVICE i AND COMM. ROUND j ON THE JETSON BOARD

	VOC, ResNet-18 size: 44.79 MB	CIFAR-100, ResNet-56 size: 2.49MB	
	$N=4,\ t=60$	$N=4,\ t=120$	$N=16,\ t=120$
$\frac{\sum_{i=1}^{N}\sum_{j=0}^{t-1}}{N(t-1)}$	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]
decode	1.63	0.32	0.33
train	55.62	74.55	29.68
encode	2.22	0.78	0.80

Round j on the Jetson board.

TABLE VII
AVERAGE PROCESSING TIMES FOR DECODING, TRAINING AND ENCODING PER CLIENT DEVICE i AND COMM. ROUND j ON THE GPU CLUSTER

	CIFAR-10 / -100		ImageNet-200			Pascal VOC	
	ResNet-20 (1.08 MB)	ResNet-56 (2.49 MB)	ResNet-50 (95.88 MB)	MobileNetV2 (10.06 MB)	EfficientNet-B0 (17.22 MB)	ResNet-18 (44.79 MB)	ViT-B/16 (343.26 MB)
$\frac{\sum_{i=1}^{N}\sum_{j=0}^{t-1}}{N(t-1)}$	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]	time_{ij}[s]
decode	0.10	0.14	1.31	0.31	0.48	0.62	2.80
train	4.03	8.38	29.23	27.03	32.41	6.27	19.62
encode	0.16	0.31	4.06	1.02	1.60	1.36	8.84

Round j on the GPU cluster.

D. Applications

Our proposed dNNC method benefits any distributed neural learning paradigm with a large number of client devices, high frequency of communication rounds (including continuous learning) or large NN architectures. All these characteristics generate data volumes that result in high communication and energy costs, or high latency due to low bandwidth, and thus a larger delay in the learning process.

In real-world applications, FL is being explored in autonomous vehicles such as self-driving cars, which generate a huge amount of data (mostly from visual sensors), which potentially leads to communication and response delays during transmission and processing on a server. Therefore, [58] investigates training models locally in the vehicle and transmitting compressed versions of parameter updates to a global server model. [59] found that the ability to share data through car-to-car communication can solve the problem of interrupted or unreliable data transfer and improve driving experience. In the health sector, FedHealth [60] was proposed as one of the first FL frameworks to accomplish accurate and personalized healthcare by aggregating patient data from wearables in a privacy-preserving manner. Furthermore, FL is used for a range of website applications, such as movie recommendation systems with >150,000 clients [61] or personalized e-commerce. For the latter, [62] presents a model that is trained in a distributed way and processes the relationships between users, videos, and products to provide videos online with ads tailored to the user and video semantics. In industry, FL is used for machinery fault diagnosis [63] to solve the problem that each client (i.e., machine) has insufficient training data and therefore cannot learn its own fault diagnosis model. For smart cities, e.g., in combination with autonomous vehicles, FL can also play an important role in predicting traffic flows [64] or improving traffic sign classification for vehicles by exploiting unlabeled smart city data [65].

VI. Conclusion

This article presented dNNC, a novel coding method for various synchronous and asynchronous distributed learning scenarios, including federated, split, and transfer learning. dNNC comprises tools especially tailored to the update data occurring in these scenarios: The parameter update tree (PUT) identifies updates of neural network partitions uniquely and provides a flexible high-level syntax interface for update management by a systems layer. The new compression and low-level coding tools — optimized structured and unstructured sparsification, federated BatchNorm folding, reversible pruning for tensor update rows, temporal context adaptation and history dependant significance coding — have been designed considering the special characteristics of the update data and have mostly been included into the 2nd edition of the NNC standard. dNNC achieves coding gains of up to 60% over NNC edition 1 and compresses updates to less than 1% of their original size without significant performance degradation of the learning system. Furthermore, dNNC saves up to 94% of the energy required for communication. In conclusion, dNNC significantly reduces the required bandwidth and energy cost for communication, and thus improves the efficiency of distributed learning.

Acknowledgment

The authors thank Tim Korjakow for setting up the Jetson Xavier NX Developer kit and for providing energy measurement functionalities on the board.

References

[1] ISO, *Information Technology – Multimedia Content Description Interface – Part 17: Compression of Neural Networks for Multimedia Content Description and Analysis*, *Internat. Org. for Standardization, Geneva, CH, Standard ISO/IEC 15938-17:2022*, 2022. [Online]. Available: https://www.iso.org/standard/78480.html

[2] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, 2020.

[3] F. Wu et al., "A survey on video action recognition in sports: Datasets, methods and applications," *IEEE Trans. Multimedia*, vol. 25, pp. 7943–7966, 2022.

[4] A. Garcia-Garcia et al., "A survey on deep learning techniques for image and video semantic segmentation," *Appl. Soft Comput.*, vol. 70, pp. 41–65, 2018.

[5] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Feb. 2021.

[6] W. Zhu, X. Wang, and W. Gao, "Multimedia intelligence: When multimedia meets artificial intelligence," *IEEE Trans. Multimedia*, vol. 22, no. 7, pp. 1823–1835, Jul. 2020.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. On Artif. Intell. And Statist.*, 2017, pp. 1273–1282.

[8] F. Sattler, A. Marban, R. Rischke, and W. Samek, "CFD: Communication-efficient federated distillation via soft-label quantization and delta coding," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2025–2038, Jul./Aug. 2022.

[9] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[10] Q. Xu, R. Zhang, Y. Zhang, Y.-Y. Wu, and Y. Wang, "Federated adversarial domain hallucination for privacy-preserving domain generalization," *IEEE Trans. Multimedia*, vol. 26, pp. 1–14, 2023.

[11] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics - Hum. Lang. Technol.*, 2010, pp. 456–464.

[12] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of deep neural networks with natural gradient and parameter averaging," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2015.

[13] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 685–693.

[14] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.

[15] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," *Proc. Mach. Learn. Syst.*, vol. 1, pp. 212–229, 2019.

[16] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4229–4238, Oct. 2020.

[17] W. Wen et al., "TernGrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1508–1518.

[18] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 560–569.

[19] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1707–1718.

[20] A. T. Suresh, X. Y. Felix, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3329–3337.

[21] S. Vargaftik et al., "EDEN: Communication-efficient and robust distributed mean estimation for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 21984–22014.

[22] R. B. Basat et al., "Quick-FL: Quick unbiased compression for federated learning," 2022, *arXiv:2205.13341*.

[23] N. Ström, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. Interspeech*, 2015, Art. no. 2015.

[24] D. Alistarh et al., "The convergence of sparsified gradient methods," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5977–5987.

[25] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4452–4463.

[26] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. Int. Conf. Learn. Representations*, 2018.

[27] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-IID data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[28] A. Mora, L. Foschini, and P. Bellavista, "Structured sparse ternary compression for convolutional layers in federated learning," in *Proc. IEEE 95th Veh. Technol. Conf.*, 2022, pp. 1–5.

[29] Y. LeCun, "The mnist database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[30] A. Malekijoo et al., "Fedzip: A compression framework for communication-efficient federated learning," 2021, *arXiv:2102.01593*.

[31] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nature Commun.*, vol. 13, no. 1, Apr. 2022, Art. no. 2032.

[32] L. Zhang, L. Shen, L. Ding, D. Tao, and L.-Y. Duan, "Fine-tuning global model via data-free knowledge distillation for non-IID federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10174–10183.

[33] D. Yao et al., "Local-global knowledge distillation in heterogeneous federated learning with non-IID data," 2021, *arXiv:2107.00051*.

[34] Z. Zhao et al., "Towards efficient communications in federated learning: A contemporary survey," *J. Franklin Inst.*, vol. 360, no. 12, pp. 8669–8703, 2023.

[35] O. R. A. Almanifi, C.-O. Chow, M.-L. Tham, J. H. Chuah, and J. Kanesan, "Communication and computation efficiency in federated learning: A survey," *Internet Things*, vol. 22, 2023, Art. no. 100742.

[36] H. Xu et al., "Compressed communication for distributed deep learning: Survey and quantitative evaluation," 2020. [Online]. Available: http://hdl.handle.net/10754/662495

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[38] C. He et al., "Fedml: A research library and benchmark for federated machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop Scalability*, 2020.

[39] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021.
[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[41] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.
[43] J. Deng et al., "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[44] M. Everingham et al., "The pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, pp. 98–136, 2015.
[45] H. Kirchhoffer et al., "Overview of the neural network compression and representation (NNR) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 5, pp. 3203–3216, May 2022.
[46] S. Wiedemann et al., "DeepCABAC: A universal compression algorithm for deep neural networks," *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 4, pp. 700–714, May 2020.
[47] W. Duan et al., "Differential weight quantization for multi-model compression," *IEEE Trans. Multimedia*, vol. 25, pp. 6397–6410, 2022.
[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[49] X. Li, M. JIANG, X. Zhang, M. Kamp, and Q. Dou, "FedBN: Federated learning on non-IID features via local batch normalization," in *Proc. Int. Conf. Learn. Representations*, 2021.
[50] M. Andreux, J. O. du Terrail, C. Beguier, and E. W. Tramel, "Siloed federated learning for multi-centric histopathology datasets," in *Proc. Domain Adapt. Representation Transfer Distrib. Collaborative Learn.: 2nd MICCAI Workshop*, 2020, pp. 129–139.
[51] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-IID data quagmire of decentralized machine learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4387–4398.
[52] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
[53] D. Becking et al., "Adaptive differential filters for fast and communication-efficient federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 3367–3376.
[54] G. Tech et al., "History dependent significance coding for incremental neural network compression," in *Proc. IEEE Int. Conf. Image Process.*, 2022, pp. 3541–3545.
[55] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 8485–8493.
[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.
[57] X. Qiu et al., "A first look into the carbon footprint of federated learning," *J. Mach. Learn. Res.*, vol. 24, no. 129, pp. 1–23, 2023.
[58] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Federated learning for data privacy preservation in vehicular cyber-physical systems," *IEEE Netw.*, vol. 34, no. 3, pp. 50–56, May/Jun. 2020.
[59] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4298–4311, Apr. 2020.
[60] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "FedHealth: A federated transfer learning framework for wearable healthcare," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 83–93, Jul./Aug. 2020.
[61] D. Neumann, A. Lutz, K. Müller, and W. Samek, "A privacy preserving system for movie recommendations using federated learning," *ACM Trans. Recomm. Syst.*, 2023, doi: 10.1145/3634686.
[62] Z.-Q. Cheng, X. Wu, Y. Liu, and X.-S. Hua, "Video ecommerce: Toward large scale online video advertising," *IEEE Trans. Multimedia*, vol. 19, no. 6, pp. 1170–1183, Jun. 2017.
[63] W. Zhang, X. Li, H. Ma, Z. Luo, and X. Li, "Federated learning for machinery fault diagnosis with dynamic validation and self-supervision," *Knowl.-Based Syst.*, vol. 213, 2021, Art. no. 106679.
[64] Y. Liu, J. J. Q. Yu, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7751–7763, Aug. 2020.
[65] A. Albaseer, B. S. Ciftler, M. Abdallah, and A. Al-Fuqaha, "Exploiting unlabeled data in smart cities using federated edge learning," in *Proc. Int. Wireless Commun. Mobile Comput.*, 2020, pp. 1666–1671.
[66] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," *A Practical Approach to Microarray Data Analysis*. Boston, MA, USA: Springer, 2003, pp. 91–109.
[67] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NeurIPS Deep Learn. Representation Learn. Workshop*, 2015.
[68] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

Daniel Becking (Graduate Student Member, IEEE) received the B.Eng. degree in microsystems technology and the M.Sc. degree in biomedical engineering from the Technical University of Berlin and HTW Berlin, Germany, in 2016 and 2020, respectively. He is currently working toward the Ph.D. degree with the Efficient Deep Learning group at the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin and the Technical University of Berlin. He is involved in international standardization activities, contributing to the ISO/IEC MPEG standard on neural network coding. His academic research interests include topics in efficient deep and distributed learning, neural compression, biomedical data and low-power hardware-software co-design.

Karsten Müller (Senior Member, IEEE) received the Dr. Ing. degree in electrical engineering and the Dipl. Ing. degree from the Technical University of Berlin, Berlin, Germany, in 2006 and 1997, respectively. Since 1997, he has been with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin, where he is currently the Head of the Efficient Deep Learning Group. His research interests include multi-dimensional video coding, efficient federated learning, and compression of neural networks. He has been involved in international standardization activities, successfully contributing to the ISO/IEC Moving Picture Experts Group for work items on visual media content description, multiview, multi-texture, 3D video coding, and neural network representation. He co-chaired an adHoc Group on 3D Video Coding from 2003 to 2012. He was the Chair and Editor of IEEE conferences and the Senior Area Editor of IEEE TRANSACTIONS ON IMAGE PROCESSING.

Paul Haase received the B.Sc. and M.Sc. degrees in electrical engineering from the Technical University of Berlin, Berlin, Germany, in 2012 and 2015, respectively. In 2013, he joined the Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin, Germany, where he is currently a Research Associate with the Video Coding & Analytics Department. He successfully contributed to the H.266/Versatile Video Coding (VVC) standardization activity of the ITU-T Moving Pictures Experts Group. His research interests include efficient representations of deep neural networks, image and video compression, and entropy coding.

Heiner Kirchhoffer received the Dipl.-Ing. (FH) degree in television technology and electronic media from the Wiesbaden University of Applied Sciences, Wiesbaden, Germany, in 2005, and the Dr.-Ing. degree from the University of Rostock, Rostock, Germany, in 2016. In 2004, he joined the Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin, Germany, where he is currently a Research Associate with the Video Coding & Analytics Department. Dr. Kirchhoffer has contributed successfully to the H.265/High Efficiency Video Coding (HEVC) and to the H.266/Versatile Video Coding (VVC) standardization activities of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Pictures Experts Group. His research interests include image and video compression, entropy coding, and efficient representations of deep neural networks.

Gerhard Tech received the Dipl.-Ing. degree in electrical engineering from RWTH Aachen University, Aachen, Germany, in 2007, and the Dr.-Ing. degree from the Technical University of Berlin, Berlin, Germany, in 2018. Since 2008, he has been with the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, Germany. His research interests mainly include compression of video, 3D video, and neural networks. He co-chaired several JCT-3 V ad hoc groups during the development of the multiview and 3D extensions of H.265/HEVC. He is the Editor of both extensions, and Software Coordinator for the 3D-HEVC and MV-HEVC reference software.

Wojciech Samek (Member, IEEE) received the Dr. rer. nat. degree with distinction (*summa cum laude*) from the Technical University of Berlin, Berlin, Germany, in 2014. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Technical University of Berlin, Berlin, Germany, and is jointly heading the Department of Artificial Intelligence and the Explainable AI Group, Fraunhofer Heinrich Hertz Institute (HHI), Berlin, Germany. He studied computer science with the Humboldt University of Berlin, Berlin, Germany, Heriot-Watt University, Edinburgh, U.K., and The University of Edinburgh, Edinburgh, U.K. He has coauthored more than 150 peer-reviewed journal and conference papers, some of them listed by Thomson Reuters as "Highly Cited Papers" (i.e., top 1%) in the field of Engineering. During his studies, he was awarded scholarships from the German Academic Scholarship Foundation and the DFG Research Training Group GRK 1589/1, and was a visiting Researcher with NASA Ames Research Center, Mountain View, USA. Dr. Samek is associated faculty, BIFOLD - Berlin Institute for the Foundation of Learning and Data, ELLIS Unit Berlin and DFG Graduate School BIOQIC, and a Member of the scientific advisory board of IDEAS NCBR. Furthermore, he is the Senior Editor of IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, an Editorial Board Member of PLoS ONE and *Pattern Recognition*, and an elected Member of the IEEE MLSP Technical Committee. He was the recipient of multiple best paper awards, including the 2020 Pattern Recognition Best Paper Award, and part of the expert group developing the ISO/IEC MPEG-17 NNC standard. He is the leading Editor of the Springer book "*Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*" in 2019, the Co-Editor of the open access Springer book "*xxAI - Beyond Explainable AI*" in 2022, and organizer of various special sessions, workshops and tutorials on topics such as explainable AI, neural network compression, and federated learning.

Heiko Schwarz received the Dipl.-Ing. degree in electrical engineering and the Dr.-Ing. degree from the University of Rostock, Rostock, Germany, in 1996 and 2000, respectively. In 1999, he joined the Fraunhofer Heinrich Hertz Institute, Berlin, Germany. Since 2010, he has been heading the Research Group Video Coding Technologies (formerly, Image and Video Coding), Fraunhofer Heinrich Hertz Institute. In October 2017, he became a Professor of image processing with the Free University of Berlin, Berlin, Germany. He has actively participated in the standardization activities of the ITU-T Video Coding Experts Group and ISO/IEC Moving Pictures Experts Group. He successfully contributed to the video coding standards H.264/AVC, H.265/HEVC, and H.266/VVC. Dr. Schwarz was a Reviewer of various international journals and international conferences. From 2016 to 2019, he was an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. He was appointed as the Co-Editor of H.264/AVC and as a Software Coordinator of the SVC reference software. He co-chaired various ad hoc groups of the standardization bodies and coordinated core experiments.

Detlev Marpe (Fellow, IEEE) received the Dipl.-Math. degree (Hons.) from the Technical University Berlin, Germany, in 1990, and the Dr.-Ing. degree from the University of Rostock, Germany, in 2004.

In 1999, he joined the Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Berlin, where he is currently the Head of the Video Communication and Applications Department. Since 1998, he has been an active participant and contributor to the development and standardization of three generations of video coding standards: H.264/AVC, H.265/ HEVC, and H.266/VVC. He is an author of more than 40 journal and more than 100 conference papers, including 6 best paper awarded publications and 10 invited papers. From 2014 to 2018, he served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS and SYSTEMS FOR VIDEO TECHNOLOGY. His research interests include still image and video coding, signal processing for communications, computer vision, machine learning, and information theory.

Dr. Marpe is inventor or co-inventor of over 1000 national patents worldwide, including more than 350 US and 100 European patents. He was a recipient of the Karl Heinz Beckurts Award in 2011, the Joseph von Fraunhofer Prize and the ITG Award, both in 2004. In 2016, he received the Best Associate Editor Award of the IEEE Circuits and Systems Society. Dr. Marpe is a member of the Association for Computing Machinery and of the Informationstechnische Gesellschaft of the Verband der Elektrotechnik Elektronik Informationstechnik e.V.

Thomas Wiegand (Fellow, IEEE) received the Dipl.-Ing. degree in electrical engineering from the Technical University of Hamburg, Hamburg, Germany, in 1995, and the Dr.-Ing. degree from the University of Erlangen-Nuremberg, Erlangen, Germany, in 2000. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Technical University of Berlin, Berlin, Germany, and is jointly heading the Fraunhofer Heinrich Hertz Institute, Berlin, Germany. As a student, he was a Visiting Researcher with Kobe University, Kobe, Japan, the University of California at Santa Barbara, Santa Barbara, CA, USA, and Stanford University, Stanford, CA, USA, where he also returned as a Visiting Professor. He was a Consultant and Co-founder of several start-up companies. Since 1995, he has been an active participant in standardization for multimedia with many successful submissions to ITU-T and ISO/IEC. Since 2018, he has been appointed the Chair of the ITU/WHO Focus Group on Artificial Intelligence for Health. For his research, he was the recipient of numerous research and innovation awards and various best paper awards for his publications. Since 2014, he named him in their list of "The World's Most Influential Scientific Minds" as one of the most cited researchers in his field. He has been elected to the German National Academy of Engineering (Acatech) and National Academy of Science (Leopoldina).