DeepHEC: Hybrid Error Coding using Deep Learning

Pablo Gil Pereira*, Andreas Schmidt**, Thorsten Herfet*
* *Telecommunications Chair*
Saarland Informatics Campus, Saarbrücken, Germany
{gilpereira, herfet}@cs.uni-saarland.de
** *Safety Engineering Department*
Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany
andreas.schmidt@iese.fraunhofer.de

***Abstract*—The distributed nature of cyber-physical systems makes reliable communication essential. Hybrid Error Coding (HEC) allows the adaptation of transmission schemes to application requirements (i.e., reliability and latency) and network conditions. However, picking an efficient HEC configuration is a computationally complex search task that must be repeated when network conditions change. In this paper, we introduce DeepHEC, a deep-learning-based approach for inferring coding configurations. Results indicate that DeepHEC is on par with search-based approaches in configuration efficiency, while significantly reducing inference time. In addition, DeepHEC decouples solution space size and inference time, thereby achieving much more predictable inference times that enable adaptive HEC on real-time systems with strict timing requirements. This is especially advantageous for cyber-physical systems that could not previously benefit from adaptive HEC.**

***Index Terms*—Cyber-Physical Systems, Error Control, Hybrid Error Coding, Deep Neural Networks**

I. Introduction

Nowadays, we see an increase in cyber-physical systems (CPS) being created for various use cases (e.g., industrial manufacturing, autonomous logistics, or energy grids). These CPS are often distributed, so reliable communication is an essential pillar of dependable system operation. Reliable communication is implemented by combining transmission schemes (i.e., ARQ or FEC) with coding approaches based on Reed Solomon [1] or Fountain Codes [2], [3]. Previous research [4] indicates that an ideal solution in terms of theoretical redundancy overhead is to use hybrid error coding (HEC). On the physical layer, HEC is often implemented as Hybrid ARQ (HARQ) [5], meaning that encoded parity packets are sent on demand. However, we use the term HEC to allow for pure FEC as well. HEC schemes have the drawback that a suitable and efficient configuration cannot be obtained from a closed-form expression but requires extensive search. This configuration must be chosen considering a) application requirements (e.g., maximum end-to-end latency or acceptable packet loss rate), which are usually constant during a system's operation, and b) communication channel parameters (e.g., packet loss rate or latency) that vary depending on the environment. A mathematical framework to find the optimal configuration is provided in [4]. Despite various optimizations of the search algorithm, sometimes the search space is too large for an efficient exploration, which makes adaptability impossible when the inference time is larger than the channel coherence time. This intractability is particularly challenging on resource-constrained, embedded devices—the natural components used to form CPS.

Machine learning has gained traction in nearly every field in recent years, thanks to advances in both hardware and available software frameworks. Machine learning algorithms can learn arbitrary functions if the architecture of the algorithm is powerful enough. When it comes to inference, deep learning algorithms have the appealing property that the computation latency is not dependent on the input values but only on the neural network architecture, its implementation, and the underlying hardware. This leads us to two research questions: Can deep neural networks be used to learn the mapping between network conditions and application requirements to HEC configurations? Can the inference task be executed with low delay and jitter, enabling its use in real-time systems that deal with changing channel conditions?

The contribution of this paper is twofold:

- We introduce DeepHEC, a deep learning approach to configure hybrid error coding systems.
- We evaluate the inference time required for both traditional search-based algorithms and our proposed deep learning algorithm, showing that learning-based approaches enable real-time adaptive reliability due to their predictably low inference time.

The remainder of the paper is structured as follows: In Section II, we discuss related work. Section III introduces the delay and loss rate models used to build a time-aware HEC scheme. In Section IV, we describe how an optimal HEC configuration can be obtained via an extensive search. A comparable implementation using deep learning is described in Section V. We evaluate both approaches in Section VI. Section VII highlights directions for future work and Section VIII concludes the paper.

II. Related Work

A. Hybrid Error Coding

Although ARQ is the dominant error correction in widely deployed protocols (i.e., TCP or QUIC), HEC has been efficiently used as well. Palmer et al. [6] use it to support partial reliability in QUIC, improving video streaming quality. Another application domain of HEC is multicast due to feedback implosion when ARQ is used with many receivers [7]. While our paper focuses on HEC with block codes, Huang et al. [8] implement a similar scheme using random linear codes (RLC) instead, thereby making the HEC delay independent of the block length at the cost of higher redundancy. The low delay of RLC codes is confirmed in [9], which studies different approaches in the context of QUIC.

B. Deep Learning in Networked Communications

In recent years, we have seen an increased interest in applying deep learning to the field of networked communications. [10] provides a recent overview of deep reinforcement learning in network applications and includes a framework to formally verify the properties of these algorithms. [11] is similar to our work in that they use traffic parameters to infer higher-level information—in their case QoS. [12] uses device-local statistics to train a predictor for end-to-end service metrics—enabling prediction-based decisions. Closest to our work is, to the best of our knowledge, [13]. However, the authors consider a joint source- and channel-coding, while we only consider channel-coding using HEC. Furthermore, their goal is to minimize error and redundancy information while we strive to fulfill an application-specific error rate, while at the same time minimizing the redundancy information.

III. Background

Timely reliability must be based on precise delay and packet loss rate models, so that the transport protocol can be configured to meet the application's delay and reliability constraints. This section introduces the models that are later used to optimally configure error control.

A. HEC Delay

HEC combines Automatic Repeat reQuest (ARQ) and Forward Error Coding (FEC). The ARQ delay is dominated by the round-trip time (RTT) required to detect a packet loss and retransmit such a packet. This retransmission process can be repeated N_C times, where N_C is the number of retransmission cycles, until either the target loss rate is achieved or the target delay expires. Under tight application delay constraints, a proactive scheme such as FEC may be a better option instead. The FEC delay is dominated by the source packet interval (T_s) required to collect k packets before encoding, where k is the block length. Consequently, the optimality of using either FEC or ARQ depends on the relation between RTT and T_s. Finally, a retransmission schedule (N_P) should be configured that determines whether the p parity packets are proactively or reactively transmitted. N_P is a vector of length $N_C + 1$ where

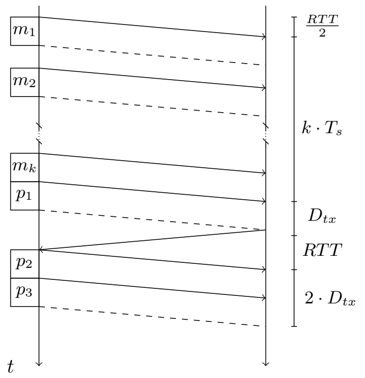


Fig. 1: HARQ delay budget. It considers the round-trip time (RTT), inter-packet time (T_s), transmission delay (D_{tx}) and block length (k).

the first position represents the FEC packets and the following ones the consecutive ARQ retransmission cycles.

$$D_{HEC}(k, N_C, N_P) = D_{FEC}(k, N_P) + D_{ARQ}(N_C, N_P) \quad (1)$$

$$D_{FEC}(k, N_P) = \frac{RTT + D_{RS}}{2} + k \cdot T_s + N_P[0] \cdot \frac{P_L}{R_C} \quad (2)$$

$$D_{ARQ}(N_C, N_P) = \sum_{c=1}^{N_C} RTT + N_P[c] \cdot \frac{P_L}{R_C} + D_{RS} + D_{PL} \quad (3)$$

The HEC delay is given in Eq. (1) and depicted in Figure 1. D_{RS} is the response delay of the system and it models operating system delays such as packet management or scheduling latencies. Although a more precise adaptation can be achieved by feeding dynamic response delays into the algorithm [14], we have opted for a rather conservative, constant value ($D_{RS} = 1\,ms$) to reduce the dimensionality of the input dataset (see Section V-A). The reactive scheme incurs the packet loss detection delay (D_{PL}), which depends on the loss detection mechanism. We assume the mechanism in [15] is implemented, which detects losses upon the arrival of 3 consecutive out-of-order packets, and therefore $D_{PL} = 4.5 \cdot T_s$. Finally, P_L and R_C correspond to the packet length and channel data rate, respectively. In the following we assume $P_L = 1500$ as this is the usual maximum MTU in IP networks.

B. HEC Packet Loss Rate

If HEC is deployed on highly dynamic systems that continuously sense the channel and immediately adapt the protocol configuration when changes are detected, the system is going to operate below the channel coherence time. Given the fast reaction to changes, the underlying channel can be modeled as a binary erasure channel (BEC) in which packet losses are i.i.d. (independent and identically distributed) and occur with probability p_e. However, the framework here presented could also be applied to channels with memory (i.e., Gilbert-Elliot channels) [16] if required at the cost of less tractable results.

$$PLR_{HEC}(k,p) = \frac{1}{k}\sum_{i=1}^{k} i \cdot Pr(I_k = i) \tag{4}$$

$$Pr(I_k = i) = \sum_{e=max(p+1,i)}^{p+i} \binom{n}{e} \cdot p_e^e \cdot (1-p_e)^{n-e} \cdot p_d\binom{e}{i} \tag{5}$$

$$p_d\binom{e}{i} = \frac{\binom{k}{i}\binom{n-k}{e-i}}{\binom{n}{e}} \tag{6}$$

The packet loss rate (PLR_{HEC}) for a BEC is given in Eq. (4), where $Pr(I_k = i)$ is the probability of i unrecoverable losses and $p_d\binom{e}{i}$ the probability that, given e erasures in a block of $n = k + p$ packets, i erasures lie within the data packets. The number of parity packets is the 1-norm of the repair schedule vector ($p = ||N_P||_1$). Note that here we assume that systematic codes are used, meaning that a verbatim copy of the original data packets is always transmitted.

IV. Optimal HEC Configuration

We say an HEC configuration (k, N_C, N_P) is optimal if, given a target delay D_T and packet loss rate PLR_T, it produces the minimum amount of redundancy information (RI, see Eq. (7)) from all possible configurations meeting both constraints. Thanks to the feedback from the receiver, the sender can stop retransmitting redundancy as soon as k packets have been correctly received, thereby reducing the overall RI. Such behavior is modeled with $p_f(c)$, which is the probability of not receiving enough packets in cycle c.

$$RI(k, N_C, N_P) = \frac{1}{k}N_P[0] + \frac{1}{k}\sum_{c=1}^{N_C} p_f(c-1) \cdot N_P[c] \tag{7}$$

$$p_f(c) = p_f(c-1) \sum_{e=n[c]-k+1}^{n[c]} \binom{n[c]}{e} \cdot p_e^e \cdot (1-p_e)^{n[c]-e} \tag{8}$$

$$n[c] = k + \sum_{b=0}^{c} N_P[b] \tag{9}$$

There is no closed-form expression to find the optimal HEC configuration under delay and reliability constraints. A solution to the optimization problem can be found using Algorithm 1, which is an adaptation of previous algorithms [4], [16] modified to execute fast enough for the generation of large datasets (see Section V-A). The search begins with the maximum number of retransmission cycles (see Eq. (10)) and block length (see Eq. (11)). $N_{c,max}$ is the maximum possible number of cycles if pure ARQ is configured, whereas k_{max} is the maximum block length when a single parity packet is transmitted in N_C retransmission cycles. For each N_C and k value pair, the number of parity packets is incremented until the PLR_T constrain is met and these are scheduled to minimize the RI (lines 8 and 9 in Algorithm 1) until the optimal configuration is found or a breaking point is reached.

Algorithm 1 HEC Configuration Search

```
Require: $D_T, PLR_T$
 1: $k^* \leftarrow 0$
 2: $N_C^* \leftarrow 0$
 3: $N_P^* \leftarrow 0$
 4: $ri^* \leftarrow \infty$
 5: $b \leftarrow 0$
 6: for $N_C = N_{C,max} \rightarrow 0$ do
 7:     for $k = k_{max} \rightarrow 1$ do
 8:         $n \leftarrow DERIVE\_N(k)$
 9:         $N_P \leftarrow GEN\_SCHEDULE(k, n, N_C)$
10:         $ri \leftarrow GET\_RI(k, N_C, N_P)$
11:         $p_r \leftarrow PLR_{HEC}(k, n-k)$
12:         $d_r \leftarrow D_{HEC}(k, N_C, N_P)$
13:         if $ri < ri^* \wedge p_r \leq PLR_T \wedge d_r \leq D_T$ then
14:             $k^* \leftarrow k$
15:             $N_C^* \leftarrow N_C$
16:             $N_P^* \leftarrow N_P$
17:             $ri^* \leftarrow ri$
18:             $b \leftarrow 1$
19:         end if
20:     end for
21:     if $b == 0$ then
22:         break
23:     end if
24:     $b \leftarrow 0$
25: end for
26: return $k^*, N_C^*, N_P^*$
```

$$N_{C,max} = \left\lfloor \frac{D_T - D_{FEC}(1,[0,1])}{D_{ARQ}(1,[0,1])} \right\rfloor \tag{10}$$

$$k_{max} = \left\lfloor \frac{D_T - \frac{RTT+D_{RS}}{2} - N_C \cdot D_{ARQ}(1,[0,1])}{T_s} \right\rfloor \tag{11}$$

The algorithm here presented can find the optimum configuration faster than a full search thanks to three assumptions:

1) The solution space is convex in N_C [16]. If an update of N_C does not reduce the RI, the algorithm stops.
2) Due to the exponential decrease in the failure probability of a cycle (see Eq. (8)), using more cycles reduces the RI. Therefore, starting from the maximum number of cycles $N_{C,max}$, in combination with the early break due to the first assumption, reduces the number of iterations.
3) Since the last retransmission cycle only occurs when all the previous ones have failed, putting parity packets at the back of the repair schedule helps reduce the RI, except when the probability of the last cycle occurring is high. Algorithm 2 efficiently checks if bringing parity packets forward in the schedule reduces the RI.

Algorithm 2 Generate Repair Schedule

```
1:  function GEN_SCHEDULE($k, n, N_C$)
2:      $p \leftarrow n - k$
3:      if $N_C == 0$ then
4:          $N_P \leftarrow [p]$
5:      else
6:          $N_P \leftarrow [0, ones(N_C - 1), p - (N_C - 1)]$
7:      end if
8:      $ri \leftarrow GET\_RI(k, N_C, N_P)$
9:      for $i = N_C \rightarrow 1$ do
10:         $N_P^{'} \leftarrow N_P$
11:         while $N_P^{'}[i] > 1$ do
12:             $N_P^{'}[i] \leftarrow N_P^{'}[i] - 1$
13:             $N_P^{'}[i-1] \leftarrow N_P^{'}[i-1] + 1$
14:             $ri^{'} \leftarrow GET\_RI(k, N_C, N_P^{'})$
15:             if $ri^{'} < ri$ then
16:                 $ri = ri^{'}$
17:                 $N_p \leftarrow N_P^{'}$
18:             else
19:                 break
20:             end if
21:         end while
22:     end for
23:     return $N_P$
24: end function
```

As shown in the Appendix, Algorithm 1 has two major overheads, namely finding the optimal number of parity packets (line 8) and the optimal repair schedule (line 9). The overall computational complexity of the algorithm is $\mathcal{O}(N_{C,max}^2 k_{max}^2 \cdot M(k_{max}, p_{max}, N_{C,max}))$, where $M(k, p, N_C) = max(kp, p^2, N_C k, N_C p)$. Such a large computational complexity has two major implications: i) the inference time may be larger than the channel coherence time, and ii) the expected inference time is highly unpredictable for different inputs. In the following we introduce DeepHEC, a deep learning approach to HEC configuration search that directly addresses these issues.

V. DeepHEC

Based on our knowledge about the search for optimal coding configurations, we have come up with a supervised deep learning approach to infer configurations. DeepHEC uses a neural network to infer k, n, and N_C, and Algorithm 2 to find the optimal repair schedule.

A. Dataset Generation

To use our supervised learning approach, it was necessary to generate a dataset. We used the following input parameters:

- Application parameters: target erasure rate, target delay, source packet interval, and packet length.
- Network parameters: channel data rate, channel erasure rate, and round-trip time.

In order to obtain a representative dataset, we have considered network traces collected in the wild for the typical network deployments (i.e., broadband [17], 4G [18], 5G [19], [20]), delay and reliability constraints for traditional applications [21] as well as more demanding applications still under development [22], and a large set of different application parameters [23] to support a wide range of real-time applications. The resulting input dataset consists of 1.200.000 samples, generated as follows: for each parameter, an order of magnitude from Table I is randomly selected with equal probability. After that, a leading number between 1 and 9 is selected randomly. Finally, Algorithm 1 is used to generate the true output labels. The resulting dataset has been divided into training/validation/test datasets with the proportion 60/20/20 and z normalization is applied before training.

Parameter	Orders of Magnitude	Unit	Reference
PLR_T	$10^{-3}, 10^{-4}, 10^{-5}$	rate	[21]
D_T	$10^0, 10^1, 10^2$	ms	[21], [22]
p_e	$10^{-2}, 10^{-3}$	rate	[19], [20]
RTT	$10^0, 10^1$	ms	[19], [20]
R_C	$10^0, 10^1, 10^2, 10^3$	Mbps	[17]–[20]
T_s	$10^{-1}, 10^0, 10^1$	ms	[23]

TABLE I: Selected orders of magnitude for the generation of the parameter dataset.

B. Model Architecture

The implemented neural network uses classification for k and p, and regression for N_C. Using classification is possible because, assuming a Vandermonde code in $GF(2^8)$ is used, $k, p \in [0, 255]$. N_P is algorithmically found once k, p, and N_C are known (see Algorithm 2). The neural network takes as input the vector $\vec{i} = [PLR_T, D_T, p_e, RTT, R_C, T_s]$ and it outputs the vector: $\vec{o} = [k, p, N_C]$. The main difference with Algorithm 1 is the number of input parameters: P_L and D_{RS} are constant values and therefore can be ignored. D_{PL} can also be omitted because it linearly depends on T_s. After the ablation study in Section VI-B, we have selected the architecture with the best accuracy (see Figure 2), consisting of 6 input neurons, 5 fully connected hidden layers with 250 neurons each and the leaky ReLU activation function, and three output layers: two with 255 outputs to perform classification for k and p, and one with a single output for N_C.

VI. Evaluation

This section evaluates the proposed training methodology for DeepHEC, including an ablation study and analysis of the different hyperparameters involved. Once a performant architecture is found, we compare its performance with Fast Search in terms of inference time and RI distribution.

A. Model Training

The models have been trained using a loss function composed of the sparse categorical cross-entropy for the classification and the mean squared error for regression. The approach in [24] has been used to find a suitable learning rate (see Figure 3a). The loss significantly drops between 10^{-3} and 10^{-4}, after which it slightly increases until 10^{-5}.

# layers	# neurons	Activation	Weight Initializer	# parameters	Accuracy		
					k	N_C	p
4	200	ReLU	He Uniform	225,113	93,72%	97,56%	99,21%
4	250	ReLU	He Uniform	318,763	98,42%	98,08%	99,79%
5	200	ReLU	He Uniform	265,313	98,09%	98,32%	99,77%
5	250	ReLU	He Uniform	381,513	99,64%	98,81%	99,96%
6	200	ReLU	He Uniform	305,513	99,22%	98,75%	99,92%
4	250	Leaky ReLU ($\alpha = 0.01$)	He Uniform	318,763	98,19%	97,97%	99,76%
5	200	Leaky ReLU ($\alpha = 0.01$)	He Uniform	265,313	97,82%	98,01%	99,73%
5	**250**	**Leaky ReLU** ($\alpha = 0.01$)	**He Uniform**	**381,513**	**99,75%**	**98,82%**	**99,94%**
6	**200**	**Leaky ReLU** ($\alpha = 0.01$)	**He Uniform**	**305,513**	**99,34%**	**98,82%**	**99,93%**
5	250	Leaky ReLU ($\alpha = 0.2$)	He Uniform	381,513	97,63%	98,23%	99,58%
6	200	Leaky ReLU ($\alpha = 0.2$)	He Uniform	305,513	97,01%	98,49%	99,55%
5	250	ELU ($\alpha = 0.5$)	He Uniform	318,763	96,26%	98,10%	99,40%
5	250	ELU ($\alpha = 1.0$)	He Uniform	318,763	89,27%	97,51%	98,25%
5	250	SELU	LeCun Normal	318,763	96,84%	98,12%	99,62%
6	200	SELU	LeCun Normal	305,513	96,62%	98,15%	99,56%

TABLE II: Neural Network hyperparameter study.

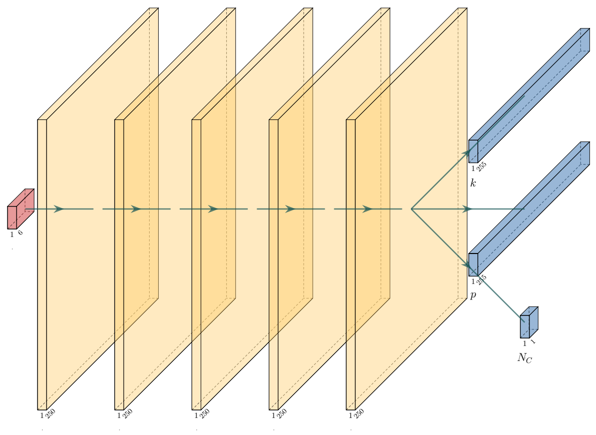


Fig. 2: DeepHEC architecture: input layer with 6 inputs, 5 fully-connected layers and three output layers with 255 outputs for k and p, and a single output for N_C.

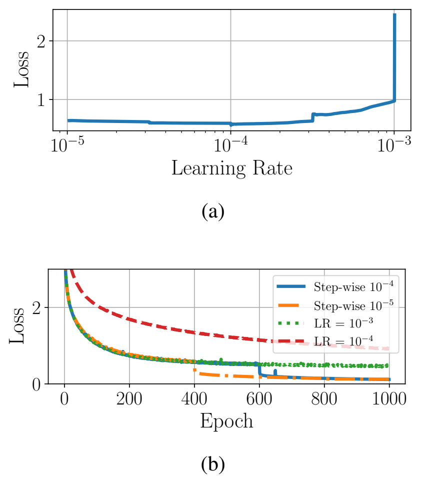


Fig. 3: Learning rate evolution for a model with 5 layers with 250 neurons each and ReLU ($\alpha = 0.001$) activation function.

Therefore, the selected maximum and minimum learning rates are 10^{-3} and 10^{-4}, respectively. These results are confirmed in Figure 3b. Fix learning rate policies either quickly converge to suboptimal solutions (10^{-3}) or require too many epochs[1] to achieve low loss (10^{-4}). In order to achieve a good trade-off between quick solution space exploration and fine-grained optimum approximation, we set the initial learning rate at 10^{-3} and reduce it once the loss has converged. Two versions of this approach are proposed: i) *Step-wise* 10^{-4}, which switches to 10^{-4} on epoch 600, and ii) *Step-wise* 10^{-5}, which first switches to 10^{-4} in epoch 400 to later switch to 10^{-5} in epoch 800. As expected, further reducing the learning rate below 10^{-4} does not produce lower loss and therefore the former approach has been selected to train all the models.

B. Hyperparameter Analysis

Table II presents the different architectures that have been tested, which were trained for 1,000 epochs with a batch size of 300. Only nonsaturating activation functions (i.e., ReLU, leaky ReLU, ELU, and SELU) have been considered to prevent vanishing and exploding gradients [25]. In order to mitigate unstable gradient problems, we used the He Uniform weight initializer for ReLU, leaky ReLU, and ELU, whereas the LeCun Normal initialization has been used for SELU, as recommended in [25]. Table III shows that there is little difference between L1 and L2 regularization. Hence, L2 regularization with a factor of 0.0001 has been selected as it finds the most valid configurations. The best performance overall is achieved with leaky ReLU ($\alpha = 0.01$) with 5 layers and 250 neurons per layer. In the following, we only consider this model, as well as leaky ReLU ($\alpha = 0.01$) with 200 neurons and 6 layers in order to analyze the performance of smaller models with slightly worse accuracy.

C. Model Performance

Now that we know that our DeepHEC approach accurately predicts the results of the extensive search, the question remains how they compare in terms of resource ef-

[1]An epoch is one complete pass through the training dataset.

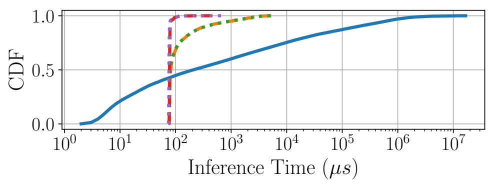
Reg.	Reg. Factor	Accuracy			Valid Configs
		k	N_C	p	
L1	0.001	99,44%	98,79%	99,81%	99,20%
L1	0.0001	99,77%	98,97%	99,96%	99,59%
L1	0.00001	99,65%	98,98%	99,98%	99,54%
L2	0.001	99,27%	98,89%	99,72%	99,39%
L2	0.0001	99,75%	98,82%	99,94%	99,59%
L2	0.00001	99,72%	98,85%	99,97%	99,52%

TABLE III: Performance impact of regularization for model with leaky ReLU ($\alpha = 0.01$), 5 layers and 250 neurons.

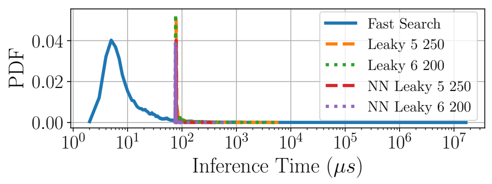
ficiency. When it comes to algorithmic complexity, DeepHEC has $\mathcal{O}(1)$ as the number of operations of a neural network is independent of the input. In contrast, Fast Search has $\mathcal{O}(N_C^2 k_{max}^2 \cdot M(k_{max}, p_{max}, N_{C,max}))$, where $M(k,p,N_C) = max(kp, p^2, N_C k, N_C p)$ (see Appendix), as the number of HEC configurations to check for optimality depends on the maximum block length (k_{max}) and number of repair cycles ($N_{C,max}$) that meet the delay constraint, as well as the necessary parity packets to ensure a loss rate below the target loss rate. Following the definition of the O-notation, DeepHEC is better—but as we are interested in performance on real-time systems, we instead care about small problem sizes and not asymptotical behavior. Therefore, we execute performance evaluations concerning inference time and redundancy information to evaluate practical performance. We do so on a desktop PC running Ubuntu 18.04 LTS on an Intel Core i7-7700 CPU at 3.6 GHz and 16GB RAM. For the search-based approach, we evaluate the algorithm proposed in Section IV (Fast Search). This algorithm is implemented in Rust[2] and compiled using `cargo build --release` to make sure the resulting code is optimized. The approach is allocation-free and only uses the Rust core and not the std library. Hence, the code can be compiled to systems without support for either of these—making it possible to run the search efficiently on embedded devices.

To compare the search-based with the learning-based approach, we have profiled both algorithms on our test dataset (cf. Section V-A) with respect to inference time and redundancy information. Figures 4a and 4b depict, respectively, the cumulative distribution function (CDF) and probability density function (PDF) of inference times for the different search algorithms. At first sight, it becomes evident that the learning-based models achieve a much more predictable inference time. When the solution space is small enough, Fast Search outperforms DeepHEC (approximately in 45% of the sample configurations). However, it also has a much longer tail, taking up to 4 orders of magnitude longer than the slowest DeepHEC inference. In nearly 20% of the cases, Fast Search achieves a delay beyond 100ms. Such a slow reaction makes adaptation impossible in fast-changing channels since the new configurations are not produced in an acceptable time. On the other hand, DeepHEC presents a much shorter tail beyond the 80th percentile, which is a result of the time spent looking for

[2]https://git.nt.uni-saarland.de/open-access/edcc2022/code



(a)



(b)

Fig. 4: Inference Time Cumulative Distribution Function (CDF) and Probability Density Function (PDF).

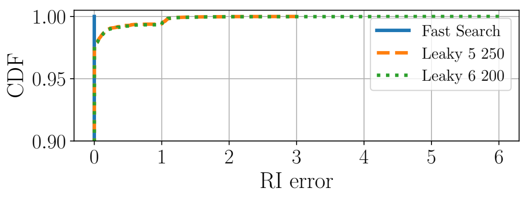


Fig. 5: Redundancy Information error Cumulative Distribution Function (CDF).

N_P. For comparison, we have included the neural network inference time that excludes the search for N_P (NN results in the figure). The worst DeepHEC inference time is in the order of single-digit milliseconds, thereby enabling quick reactions to channel changes thanks to the decoupling between solution space size and inference time.

Figure 5 shows the CDF for the RI error, i.e., how much the RI of a model deviates from the optimal RI found by Fast Search. Both models are able to find valid configurations in 99.3% of the cases so the slightly better accuracy of the larger model with 5 layers and 250 neurons does not result in a significant RI improvement, although its RI error has a shorter tail. However, further reductions in the model size come at the cost of a drop in the number of valid configurations (see Table IV) and thus other mechanisms to reduce the model size without a negative impact on the accuracy should be explored in the future, which are discussed in Section VII.

Finally, we executed a microbenchmark on a "denoised" Raspberry Pi 4 system (minimal Raspian OS running only SSH and our evaluation code) to support our claim about algorithmic complexity. We picked several inputs so that their inference times span several orders of magnitude and measured cycles using Linux `perf` tools. The result is that for DeepHEC, all cycle counts are in the same order of magnitude (around 2.5×10^6), while for Fast Search they span several orders of magnitude (10^6 to beyond 10^9).

Model	Accuracy			Valid Configurations
	k	N_C	p	
Leaky 5 200	97,82%	98,01%	99,73%	99,01%
Leaky 5 250	99,75%	98,82%	99,94%	99,34%
Leaky 6 200	99,34%	98,82%	99,93%	99,35%

TABLE IV: Percentage of valid configurations for different architectures.

VII. Future Work

Although DeepHEC heavily reduces the inference time, the results presented in this paper show that Fast Search can outperform DeepHEC in roughly 45% of the cases. Therefore, it is still an open research question whether these algorithms can be deployed on embedded devices (either via hardware acceleration or more efficient neural network architectures) or dedicated, algorithmic-based solutions will still dominate embedded deployments. In future work, we intend to look into more efficient architectures towards embedded DeepHEC. We believe most of the model complexity comes from the fact that the output parameters are quantized, so that slight changes in the input may produce large output changes, resulting in discontinuities that require larger neural networks to be learned. Therefore, we plan to relax the RI-optimality constraint to reduce the model size further and, at the same time, define an RI-based loss to nudge the solution towards (suboptimal) valid configurations that do not produce a large RI increase. Finally, for DeepHEC to be deployable on embedded devices with limited memory and computational power, model quantization, and pruning are interesting alternatives to explore in order to heavily reduce the model size and inference time.

VIII. Conclusion

This paper introduces DeepHEC, a deep-learning-based approach to finding optimal configurations for HEC schemes. We have evaluated its efficiency (in terms of redundancy information) as well as latency (in terms of absolute inference time as well as inference time predictability)—showing that DeepHEC can compete with existing search-based algorithms and can decrease inference times significantly. Thanks to its more predictable inference time, DeepHEC can bring hybrid error coding to cyber-physical systems that require a real-time operation in order to meet the application requirements, so that these systems can benefit from adapting their coding scheme—a task that was previously impossible to complete.

IX. Acknowledgments

The work is supported by the German Research Foundation (DFG) as part of SPP 1914 "Cyber-Physical Networking" under grant 315036956. We would like to thank Moritz Miodek for his support in the implementation of Fast Search.

References

[1] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM computer communication review*, vol. 27, no. 2, pp. 24–36, 1997.

[2] D. J. MacKay, "Fountain codes," *IEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[3] A. Shokrollahi, "Raptor codes," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[4] M. Gorius, Y. Shuai, and T. Herfet, "Predictably reliable media transport over wireless home networks," in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 2012, pp. 62–67.

[5] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The next generation wireless access technology*. Academic Press, 2020.

[6] M. Palmer, T. Krüger, B. Chandrasekaran, and A. Feldmann, "The QUIC fix for optimal video streaming," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2018, pp. 43–49.

[7] D. Rubenstein, J. Kurose, and D. Towsley, "A study of proactive hybrid FEC/ARQ and scalable feedback techniques for reliable, real-time multicast," *Computer Communications*, vol. 24, no. 5-6, pp. 563–574, 2001.

[8] Y.-z. Huang, S. Mehrotra, and J. Li, "A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming," in *2009 IEEE International Conference on Multimedia and Expo*. IEEE, 2009, pp. 718–725.

[9] F. Michel, Q. De Coninck, and O. Bonaventure, "QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–9.

[10] T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira, "Verifying learning-augmented systems," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 305–318.

[11] S. Xiao, D. He, and Z. Gong, "Deep-Q: Traffic-driven QoS Inference using Deep Generative Network," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 67–73.

[12] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 672–698, 2017.

[13] N. Farsad, M. Rao, and A. Goldsmith, "Deep learning for joint source-channel coding of text," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 2326–2330.

[14] A. Schmidt, S. Reif, P. G. Pereira, T. Honig, T. Herfet, and W. Schroder-Preikschat, "Cross-layer pacing for predictably low latency," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 1–6.

[15] B. Oh, J. Han, and J. Lee, "Timer and sequence based packet loss detection scheme for efficient selective retransmission in DCCP," in *International Conference on Future Generation Communication and Networking*. Springer, 2010, pp. 112–120.

[16] M. Gorius, "Adaptive delay-constrained internet media transport," 2012.

[17] Federal Communications Commission. (2020) Raw Data - Measuring Broadband America. https://www.fcc.gov/oet/mba/raw-data-releases.

[18] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proceedings of the 9th ACM multimedia systems conference*, 2018, pp. 460–465.

[19] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang, "A first look at commercial 5G performance on smartphones," in *Proceedings of The Web Conference 2020*, 2020, pp. 894–905.

[20] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma, "Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 479–494.

[21] "ITU-T Y.1541: Network performance objectives for IP-based services," ITU, Tech. Rep., December 2011.

[22] "The Tactile Internet - ITU-T Technology Watch Report," ITU, Tech. Rep., August 2014.

[23] Youtube. (2021) Recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en.

[24] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017, pp. 464–472.

[25] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

Appendix

Lemma A.1: The binomial coefficient $\binom{n}{k}$, when implemented with dynamic programming, requires k operations.

The lemma above directly follows from the fact that any bonimial coefficient fulfills $\binom{n}{k} = \binom{n}{n-k}$ and can be calculated as follows

$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n-i+1}{i}$$

Lemma A.2: Given the block length k and number of parity packets p, the complexity of $PLR_{HEC}(k,p)$ (Eq. (4)) is $C_{PLR}(k,p) = \mathcal{O}(max(k^3, kp^2))$.

If exponentiation by squaring is used, the power of a number x^n has complexity $\mathcal{O}(log_2(n))$. As a result, the complexity of Eq. (4) can be obtained as follows

$$\underbrace{k}_{\text{outer sum}} \cdot \underbrace{max(p,i)}_{\text{inner sum}}(\underbrace{\mathcal{O}(log_2(e\cdot(n-e)))}_{\text{powers}} + \underbrace{\mathcal{O}(i) + \mathcal{O}(e-i)}_{\text{binomial coefficients}})$$

Since $i \leq k$ and $e \leq p+i$, the computational complexity of Eq. (4) is

$$C_{PLR}(k,p) = \mathcal{O}(max(k^3, kp^2)) \tag{12}$$

Lemma A.3: Given an HEC scheme with block length k, number of parity packets p and number repair cycles N_C, the computational complexity of calculating the RI (Eq. (7)) is $C_{RI}(k,p,N_C) = \mathcal{O}(max(N_C^2k^2, N_C^2kp))$.

Assume the result of evaluating Eq. (9) $\forall\, c \in [1, N_C]$ is stored in a vector to avoid evaluating the same expression several times. As a result, the complexity of Eq. (8) can be obtained as follows

$$\underbrace{c}_{\text{recursivity}} \cdot \underbrace{k}_{\text{sum}} \cdot (\underbrace{\mathcal{O}(e)}_{\text{binomial coefficient}} + \underbrace{\mathcal{O}(log_2(e\cdot(n[c]-e)))}_{\text{powers}} + \underbrace{\mathcal{O}(1)}_{\text{Eq. (9)}})$$

Since $c \leq N_C$, $e \leq n[c]$ and $n[c] \leq k+p$, the complexity expression above results in $\mathcal{O}(max(N_Ck^2, N_Ckp))$. Therefore, the complexity of Eq. (7) is

$$C_{RI}(k,p,N_C) = \mathcal{O}(max(N_C^2k^2, N_C^2kp)) \tag{13}$$

Lemma A.4: Given $N_{C,max}$ the maximum number of retransmission cycles (Eq. (10)), k_{max} the maximum block length (Eq. (11)), and p_{max} the optimal number of parity packets for the block length k_{max}, the computational complexity of finding the optimal codeword length n is $\mathcal{O}(N_{C,max}k_{max}^2p_{max} \cdot max(k_{max}^2, p_{max}^2))$.

The codeword length n is looked for in line 8 of Algorithm 1, which increases p starting from $p = 0$ until $p_{opt}(k)$ is found for each $k \in [1, k_{max}]$, such that $PLR_{HEC}(k, p_{opt}(k)) \leq PLR_T$. The optimal number of parity packets is a monotonically increasing function since, for increasing k and constant p, the redundancy information decreases ($\frac{p}{k+1} < \frac{p}{k}$), and therefore $PLR_{HEC}(k+1,p) \geq PLR_{HEC}(k,p)$. As a result, the optimal number of parity packets fulfills that $p_{opt}(k) \leq p_{max}\,\forall\, k \in [1, k_{max}]$ where $p_{max} = p_{opt}(k_{max})$.

$$\underbrace{N_{C,max}}_{\substack{\text{Line 6}\\ \text{Algorithm 1}}} \cdot \underbrace{k_{max}}_{\substack{\text{Line 7}\\ \text{Algorithm 1}}} \cdot \underbrace{p_{max}}_{\substack{\text{Increase } p\\ \text{until } p_{opt}(k)}} \cdot \underbrace{\mathcal{O}(max(k_{max}^3, k_{max}p_{max}^2))}_{PLR_{HEC}(k,p)\leq PLR_T \text{ check}}$$

Lemma A.5: Given $N_{C,max}$ the maximum number of retransmission cycles (Eq. (10)), k_{max} the maximum block length (Eq. (11)), and p_{max} the optimal number of parity packets for the block length k_{max}, the computational complexity of finding the optimal repair schedule is $\mathcal{O}(N_{C,max}^2k_{max}^2 \cdot M(k_{max}, p_{max}, N_{C,max}))$, where $M(k,p,N_C) = max(kp, p^2, N_Ck, N_Cp)$.

Every time a parity packet is brought forward in the repair schedule (see lines 12 to 20 in Algorithm 2), only the failure probability of the two modified cycles changes (see Eq. (8)). For an efficient implementation of Algorithm 2, our implementation maintains an array of length N_C whose entries are the failure probability of each cycle, $p_f(c)\,\forall\, c \in [1, N_C]$, such that only the modified failure probabilities are recalculated, thereby achieving the following complexity:

$$\underbrace{C_{RI}(k,p,N_C)}_{\text{Line 8 Algorithm 2}} + \underbrace{N_C}_{\substack{\text{Line 9}\\ \text{Algorithm 2}}} \cdot \underbrace{p}_{\substack{\text{At max. } p\\ \text{packets}\\ \text{forward}}} \cdot \underbrace{\mathcal{O}(max(k^2, kp))}_{\text{Recalculate RI}}$$

Note that all operations with complexity $\mathcal{O}(1)$ are not included, as the RI and schedule change clearly dominate. Bear in mind that, for the recalculation of the RI, only the summation in Eq. (8) needs to be calculated for two different cycles, resulting in the complexity $\mathcal{O}(k^2, kp)$. Eq. (14) shows the final complexity of Algorithm 2.

$$C_{Gen\_N_p}(k,p,N_C) = \mathcal{O}(N_Ck \cdot M(k,p,N_C)) \tag{14}$$

$$M(k,p,N_C) = max(kp, p^2, N_Ck, N_Cp)$$

Since $N_C \leq N_{C,max}$, $k \leq k_{max}$ and $p \leq p_{max}$, and accounting for the iterations over $N_{C,max}$ and k_{max} in Algorithm 1, then Lemma A.5 is proven.

Lemma A.6: Given $N_{C,max}$ the maximum number of retransmission cycles (Eq. (10)), k_{max} the maximum block length (Eq. (11)), and p_{max} the optimal number of parity packets for the block length k_{max}, the computational complexity of Fast Search (Algorithm 1) is $\mathcal{O}(N_{C,max}^2k_{max}^2 \cdot M(k_{max}, p_{max}, N_{C,max}))$, where $M(k,p,N_C) = max(kp, p^2, N_Ck, N_Cp)$.

This lemma directly follows from Lemma A.4 and Lemma A.5. In practice, at least one of the parameters (k_{max}, p_{max} and $N_{C,max}$) may dominate the others, depending on the relation between channel parameters and application constraints. In order to better analyze those cases, the complexity of finding the optimal codeword length should be considered.