LightTune: Lightweight Online Fine-Tuning for 6G

Ramy E. Ali and Federico Penna
Samsung Semiconductor, San Diego, CA 92121, USA
ramy.ali@samsung.com, f.penna@samsung.com

***Abstract*—Machine learning (ML) models for mobile devices are often trained offline and deployed in environments that may differ from their training conditions, leading to performance degradation even when real-world data is used for training. To address this training-test mismatch, we propose a lightweight, backpropagation-free online fine-tuning framework termed LightTune that refines the ML models using live test-time data. This online fine-tuning mechanism is opportunistically triggered when the model's performance falls below a desired threshold, ensuring minimal computational overhead and efficient responsiveness.**

As a practical demonstration, we integrate LightTune into a block error rate (BLER) prediction algorithm for 6G mobiles. This integration enables the model to dynamically adapt to previously unseen channel conditions in real-time. Simulation results show a substantial reduction in the average BLER prediction error up to 43.5% with online fine-tuning. Furthermore, we leverage this BLER prediction algorithm for link adaptation and demonstrate average throughput improvements of up to 9.5% compared to a conventional table-based outer loop link adaptation algorithm.

I. Introduction

Machine learning (ML)-based wireless algorithms are emerging as a promising direction for 6G applications, including channel state information (CSI) prediction and beam management [1]. However, a major bottleneck in realizing their full potential lies in the training-test mismatch. Typically, these ML models are trained offline using synthetic data, which often fails to capture the complexity and variability of real-world environments. Even when real data is used, dynamic channel conditions can lead to significant performance degradation after deployment [2], [3].

To address this training-test mismatch challenge, the ML models need to adapt to the deployment dynamic environment through online learning [4], [5]. Online learning, however, may not be feasible for resource-constrained devices such as a user equipment (UE) that may not support gradient computations and backpropagation in real-time. In this work, we propose a lightweight online fine-tuning framework that enables the UE to incrementally refine its ML model using inference-time (test-time) observations. Our framework is suitable for prediction tasks where the ground-truth of the predicted metric becomes available after a delay, such as in link-level throughput estimation or block error rate (BLER) prediction. Our approach leverages this delayed ground-truth data to perform targeted updates to the ML model, thereby improving the model performance over time. Unlike conventional learning methods that rely on backpropagation [3], [6], [7], our approach is backpropagation-free, making it suitable for deployment on UEs with limited memory and computational power. Specifically, our contributions are as follows.

1) We develop a fine-tuning algorithm termed as LightTune that is opportunistically triggered when the performance of the ML model, that is initially trained offline, is not as desired. Our proposed online fine-tuning algorithm offers (a) *backpropagation-free* online fine-tuning by applying the forward-forward (FF) algorithm [8], enhanced by a *newly proposed loss function with closed-form gradients*, which enables low-complexity online gradient computation, and (b) *buffer-less* fine-tuning through a proposed threshold-based update policy that decides which samples are used on a sample-by-sample basis, without storing them in an experience replay buffer first [9]–[11]. To the best of our knowledge, LightTune is the first application of the FF algorithm in wireless communications.
2) As a practical use case of LightTune, we develop a BLER prediction algorithm for 6G systems. With LightTune, substantial improvements in BLER prediction accuracy are achieved. Leveraging this enhanced BLER prediction for link adaptation yields notable throughput gains compared to conventional outer loop table-based baselines [12].

Next, we discuss background concepts and prior works in Section II. LightTune is then presented in Section III and is applied to BLER prediction in Section IV. Finally, we present our simulation results in Section V.

II. Background and Related Works

A. The Forward-Forward (FF) Algorithm

In contrast to the usual ML training using backpropagation, the FF algorithm trains each layer locally without backpropagation. This offers the advantage of avoiding the complex operations of propagating the error derivatives backward through the layers. Hence, it is appealing when used for resource-constrained devices such as UEs that may not support backpropagation. The FF algorithm is primarily used for image classification problems and is based on two main steps [8].

1) **Forward Pass with Positive Data Samples**
A positive data sample is defined as the tuple $(\boldsymbol{x}, y_+)$, where $\boldsymbol{x} \in \mathbb{R}^d$ is the input d-dimensional feature vector and $y_+ \in \mathcal{Y} = \{y_1, \ldots, y_C\}$ is the corresponding true label. The positive pass operates on the positive data and adjusts the model parameters with the goal of increasing a goodness value in every layer above a pre-defined threshold T.
2) **Forward Pass with Negative Data Samples**
A negative data sample is defined as the tuple $(\boldsymbol{x}, y_-)$, where $y_- \in \mathcal{Y} \setminus \{y_+\}$ is an incorrect label. The negative pass operates on negative data and adjusts the model to decrease a goodness value in every layer below the threshold T.

In this work, we construct the i-th positive data sample as $\boldsymbol{x}_{+}^{(i)} = [\boldsymbol{x}^{(i)}, y_{+}^{(i)}]^{\top} \in \mathbb{R}^{d+1}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^{d}$ is the i-th input feature vector and $y_{+}^{(i)} \in \mathcal{Y}$ is the corresponding true label. The corresponding negative sample is constructed by pairing the same input $\boldsymbol{x}^{(i)}$ with an incorrect label $y_{-}^{(i)} \in \mathcal{Y} \setminus \{y_{+}^{(i)}\}$, and is denoted as $\boldsymbol{x}_{-}^{(i)} = [\boldsymbol{x}^{(i)}, y_{-}^{(i)}]^{\top}$. Next, we describe the goodness function, the loss function and the inference.

- **Goodness Function.** We consider a ML model with parameters $\boldsymbol{\theta}$ and L layers. The goodness is defined as the sum of the squares of the activations of the output layer L [8]. It represents how well the model responds to an input.
  For positive samples (i.e., inputs paired with correct labels), the goodness should be high, whereas for negative samples (i.e., inputs paired with incorrect labels), the goodness should be low. The goodness of a label $y \in \mathcal{Y}$ when it is associated with a feature vector $\boldsymbol{x}$ is computed as

$$G(\boldsymbol{x}, y) = \left\| f_{\boldsymbol{\theta}}^{(L)}(\boldsymbol{x}, y) \right\|_2^2, \tag{1}$$

  where $f_{\boldsymbol{\theta}}^{(l)}(\boldsymbol{x}, y)$ denotes the output of layer $l \in \{1, 2, \ldots, L\}$ corresponding to the sample $[\boldsymbol{x}, y]^{\top}$.
- **Softplus Loss Function.** The FF algorithm typically employs the Softplus loss function [13] to push the goodness of the positive samples above the threshold T, and the goodness of the negative samples below the threshold T.
  We introduce a few notations before defining the Softplus loss function. The outputs of layer $l \in \{1, 2, \ldots, L\}$ for the i-th positive and negative samples are denoted by $\boldsymbol{h}_{+,l}^{(i)}$ and $\boldsymbol{h}_{-,l}^{(i)}$, respectively. That is, we have

$$\boldsymbol{h}_{+,l}^{(i)} = f_{\boldsymbol{\theta}}^{(l)}(\boldsymbol{x}_{+}^{(i)}), \boldsymbol{h}_{-,l}^{(i)} = f_{\boldsymbol{\theta}}^{(l)}(\boldsymbol{x}_{-}^{(i)}). \tag{2}$$

  The positive and negative goodness of layer l corresponding to the i-th sample are then defined as

$$g_{+,l}^{(i)}[j] = \left(h_{+,l}^{(i)}[j]\right)^2, g_{-,l}^{(i)}[j] = \left(h_{-,l}^{(i)}[j]\right)^2, \tag{3}$$

  for a neuron index $j \in \{1, 2, \ldots, M_l\}$, where M_l denotes the total number of neurons in layer l. The positive and negative goodness vectors for layer l are then given as $\boldsymbol{g}_{+,l}^{(i)} = \left[g_{+,l}^{(i)}[1], g_{+,l}^{(i)}[2] \ldots g_{+,l}^{(i)}[M_l]\right]^{\top}$ and $\boldsymbol{g}_{-,l}^{(i)} = \left[g_{-,l}^{(i)}[1], g_{-,l}^{(i)}[2] \ldots g_{-,l}^{(i)}[M_l]\right]^{\top}$. For sets of positive and negative samples denoted by $\mathcal{S}_{+}$ and $\mathcal{S}_{-}$, the Softplus loss for layer l is given as

$$\mathcal{L}_{\text{Softplus},l} = \frac{1}{|\mathcal{S}_{+}|} \sum_{i \in \mathcal{S}_{+}} \frac{1}{M_l} \sum_{j=1}^{M_l} \ln\left(1 + e^{-\left(g_{+,l}^{(i)}[j] - T\right)}\right) + \frac{1}{|\mathcal{S}_{-}|} \sum_{i \in \mathcal{S}_{-}} \frac{1}{M_l} \sum_{j=1}^{M_l} \ln\left(1 + e^{\left(g_{-,l}^{(i)}[j] - T\right)}\right). \tag{4}$$

- **Inference.** Given a feature vector $\boldsymbol{x}$, the predicted label $\hat{y}$ in the FF algorithm is computed by selecting the label $y \in \mathcal{Y}$ whose associated goodness is maximal as follows

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} G(\boldsymbol{x}, y). \tag{5}$$

Remark 1 (Output layer in the FF algorithm). *Unlike conventional classifiers, where the output layer's dimension is equal to the number of classes C, the output layer in the FF algorithm may have a number of neurons not equal to C. This is because the FF algorithm predicts the label based on goodness, as in Fig. 1.*

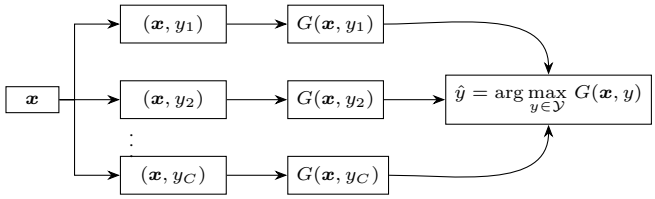


Fig. 1: Inference process in the FF algorithm: the input $\boldsymbol{x}$ is paired with each candidate label $y \in \mathcal{Y}$ and the label with highest goodness is selected.

B. Experience Replays

Experience replay is used in reinforcement learning (RL) and continual learning to reduce sample correlation and mitigate forgetting by storing and randomly replaying past experiences [9]–[11] . While the experience replay approach in [9] treats all experiences equally, the prioritized experience replay method [10] samples the surprising experiences more frequently. The surprise of a sample is quantified in terms of the error between the predicted and actual reward.

LightTune is inspired by this prioritized replay approach in that it leverages the *surprising* samples for online fine-tuning. While a straightforward implementation of online fine-tuning might rely on a replay buffer, LightTune instead dynamically identifies informative samples with high prediction error on a per-sample basis and uses them immediately for fine-tuning.

C. Related Works

The problem of adapting ML models after deployment under real-world practical constraints such as avoiding backpropagation has received limited attention in the literature. The prior works can be classified into three categories as follows.

1) **Offline Training.** Several prior works adopt an offline training paradigm, where the ML model is trained once and its parameters remain fixed after deployment [14], [15]. While simple, this approach suffers from a training-test mismatch, often resulting in significantly degraded inference performance in dynamic environments.
2) **Offline Training with Online Calibration/Adaptation.** To mitigate the effects of training-test mismatch without modifying the deployed ML model, calibration techniques have been proposed in [16], [17]. These methods utilize acknowledgment (ACK) and negative acknowledgment (NACK) signals of physical downlink shared channel (PDSCH) data transmissions to adjust the model's output. Importantly, the model parameters remain unchanged.
3) **RL.** While many works explored deep RL solutions [3], [6], [7], they overlook the practical constraints of UE lacking the

computational and storage resources of performing periodic gradient computation to update the ML model.

Our work. In contrast, our approach opportunistically fine-tunes the ML model only as needed, enabling real-time adaptation to changing data distributions.

III. Proposed Fine-tuning Algorithm: LightTune

We propose a closed-loop learning framework that enables adaptive refinement of a deployed ML model, initially trained offline, when its performance deviates from desired behavior. Initially, the FF algorithm is used to train a baseline ML model offline. Subsequently, the FF algorithm is employed for online fine-tuning of the baseline ML model. This mechanism is feasible in scenarios where ground-truth labels can be inferred at a later time. In wireless communication systems, such ground-truth labels can be derived from key performance indicators (KPIs), including BLER, bit error rate (BER) and throughput.

A. Proposed Loss: Quadratic Softplus Approximation Loss

To avoid the computational complexity introduced by exponentiation and division in the Softplus loss, we propose an alternative loss function that is simpler while achieving comparable performance. For layer l, the proposed quadratic loss is defined as

$$\mathcal{L}_{\text{Prop},l} = \frac{1}{|\mathcal{S}_+|}\sum_{i\in\mathcal{S}_+}\frac{1}{M_l}\sum_{j=1}^{M_l}\Big((g_{+,l}^{(i)}[j]-T)^2 - 4(g_{+,l}^{(i)}[j]-T)\Big) + \frac{1}{|\mathcal{S}_-|}\sum_{i\in\mathcal{S}_-}\frac{1}{M_l}\sum_{j=1}^{M_l}\Big((g_{-,l}^{(i)}[j]-T)^2 + 4(g_{-,l}^{(i)}[j]-T)\Big). \quad (6)$$

Theoretical Insights on the Proposed Loss. We derive our alternative loss from the second-order Taylor expansion of the Softplus function around $x=0$, $f(x)=\ln(1+e^x) = \ln 2 + \frac{1}{2}x + \frac{1}{8}x^2 + R_2(x)$, where $R_2(x) = \frac{f^{(3)}(\xi)}{3!}x^3$ for some $\xi\in[0,x]$. By applying this expansion to the Softplus loss and scaling, we obtain our proposed loss.

B. Gradient Computations

Based on this proposed loss function, for a multilayer perceptron (MLP), we can compute the gradient of layer l. Denote the weight matrix and bias of layer l by $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$, respectively. Our fine-tuning algorithm just uses one positive sample $\boldsymbol{x}_+ = [\boldsymbol{x}, y_+]^\top$ and one negative sample $\boldsymbol{x}_- = [\boldsymbol{x}, y_-]^\top$, where $y_-\in\mathcal{Y}\setminus\{y_+\}$. Hence, $|\mathcal{S}_+| = |\mathcal{S}_-| = 1$. Then, we can compute the derivatives of the loss assuming rectified linear unit (ReLU) activations as follows

$$\begin{aligned}\nabla_{\boldsymbol{W}_l}\mathcal{L}_{\text{Prop},l} &= (2(\boldsymbol{g}_{+,l}-T)-4)\odot 2\boldsymbol{h}_{+,l}\odot\mathbf{1}(\boldsymbol{x}_{+,l}>0)\cdot\boldsymbol{x}_{+,l}^\top \\ &\quad + (2(\boldsymbol{g}_{-,l}-T)+4)\odot 2\boldsymbol{h}_{-,l}\odot\mathbf{1}(\boldsymbol{x}_{-,l}>0)\cdot\boldsymbol{x}_{-,l}^\top, \\ \nabla_{\boldsymbol{b}_l}\mathcal{L}_{\text{Prop},l} &= (2(\boldsymbol{g}_{+,l}-T)-4)\odot 2\boldsymbol{h}_{+,l}\odot\mathbf{1}(\boldsymbol{x}_{+,l}>0) \\ &\quad + (2(\boldsymbol{g}_{-,l}-T)+4)\odot 2\boldsymbol{h}_{-,l}\odot\mathbf{1}(\boldsymbol{x}_{-,l}>0), \end{aligned} \quad (7)$$

where $\odot$ denotes element-wise multiplication (Hadamard multiplication) and the indicator function $\mathbf{1}(\boldsymbol{x}>0)$ is applied element-wise. Since we have only one positive and one negative sample, the gradient computation and the weight update are performed just once at each fine-tuning step.

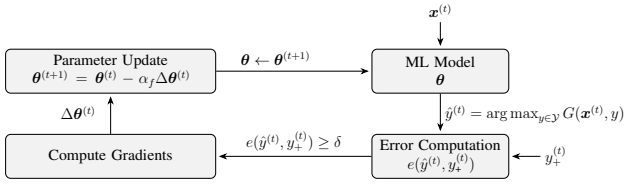


Fig. 2: The proposed fine-tuning algorithm, LightTune, uses the delayed true label to compute the prediction error and update the model if needed.

C. Threshold-based Fine-tuning Without Replay Buffer

To avoid having a buffer at the UE, LightTune adopts a threshold-based update policy. Specifically, the model is fine-tuned in a sample-by-sample manner without retaining past samples. The *surprise* of a sample at time t is quantified by the prediction error between the model's predicted label $\hat{y}^{(t)}$ and the true label $y_+^{(t)}$. This error, denoted as $e(\hat{y}^{(t)}, y_+^{(t)})$, can be computed using various metrics such as absolute error, mean squared error (MSE), etc. When the error reaches a pre-defined threshold δ, the sample $[\boldsymbol{x}^{(t)}, y_+^{(t)}]^\top$ is treated as a positive sample for fine-tuning. A corresponding negative sample $[\boldsymbol{x}^{(t)}, y_-^{(t)}]^\top$ is constructed by pairing the same input $\boldsymbol{x}^{(t)}$ with an incorrect label $y_-^{(t)}$ that can be selected uniformly at random from the set $\mathcal{Y}\setminus\{y_+^{(t)}\}$ or it can be selected as the incorrect model prediction $y_-^{(t)} = \hat{y}^{(t)}$ (i.e., hard sampling) [18]. This procedure is provided in Alg. 1 and Fig. 2.

Algorithm 1 Proposed Fine-tuning Algorithm: LightTune

Input: Current model parameters $\boldsymbol{\theta}^{(t)}$, fine-tuning threshold δ, Adam optimizer parameters $(t_{\text{A}}, \beta_1, \beta_2, \epsilon)$, fine-tuning learning rate α_f
Output: ML prediction $\hat{y}^{(t)}$, fine-tuned model parameters $\boldsymbol{\theta}^{(t+1)}$

1: Extract feature vector $\boldsymbol{x}^{(t)}$ and retrieve actual metric $y_+^{(t)}$
2: Compute model prediction: $\hat{y}^{(t)} = \arg\max_{y\in\mathcal{Y}} G\left(\boldsymbol{x}^{(t)}, y\right)$
3: **if** $e(\hat{y}^{(t)}, y_+^{(t)}) \geq \delta$ **then**
4: Construct positive sample: $\boldsymbol{x}_+^{(t)} = [\boldsymbol{x}^{(t)}, y_+^{(t)}]^\top$
5: Construct negative sample: $\boldsymbol{x}_-^{(t)} = [\boldsymbol{x}^{(t)}, y_-^{(t)}]^\top$
6: Compute gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Prop}}^{(t)}$ using $\boldsymbol{x}_+^{(t)}$ and $\boldsymbol{x}_-^{(t)}$ (as in (7))
7: Compute updated model parameters as follows [19],

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha_f\cdot\frac{\boldsymbol{m}_t/(1-\beta_1^{t_{\text{A}}})}{\sqrt{\boldsymbol{v}_t/(1-\beta_2^{t_{\text{A}}})}+\epsilon} \quad \text{(element-wise)}$$

8: **end if**

D. Fine-tuning Variants and Convergence

While LightTune is compatible with any optimizer, we focus on two variants based on Adam optimizer [19]. Adam is a widely used optimizer that adapts each parameter by maintaining two running averages (moments): the first moment (mean of gradients) at time t denoted by $\boldsymbol{m}_t$ and the second moment (variance of gradients) at time t denoted by $\boldsymbol{v}_t$.

1) **Standard Adam Update.** Each fine-tuning step $t\in\{1,2,\dots\}$ reuses the moments from the previous step $t-1$, with the Adam time step t_{A} set as $t_{\text{A}} = t$. That is,

$$\begin{aligned}\boldsymbol{m}_t &= \beta_1\boldsymbol{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Prop}}^{(t)}, \\ \boldsymbol{v}_t &= \beta_2\boldsymbol{v}_{t-1} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Prop}}^{(t)})^2, \end{aligned} \quad (8)$$

where β_1 and β_2 are parameters of Adam optimizer and $(.)^2$ is applied element-wise.

2) **One-Step Update.** In this variant, at fine-tuning step t, Adam's time step is reset to $t_{\text{A}} = 1$ and the moments are reinitialized ($\boldsymbol{m}_{t-1} = \boldsymbol{m}_0 = \mathbf{0}, \boldsymbol{v}_{t-1} = \boldsymbol{v}_0 = \mathbf{0}$),

$$\boldsymbol{m}_t = (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{Prop}^{(t)}}, \boldsymbol{v}_t = (1-\beta_2)(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(t)}_{\text{Prop}})^2. \quad (9)$$

This variant is memory-efficient in the sense that the moments are not stored across the fine-tuning steps.

Remark 2 (**Rationale for Moments Reset**). *In LightTune's online, sparse-update setting, "surprising" samples may signal a distribution shift. The one-step variant resets moments, making the update rely solely on the current gradient (becoming a normalized gradient step), enabling faster real-time adaptation.*

Theorem 1 (**Convergence**). *Assuming (1) boundedness of the data and the model, (2) at least one active neuron in the final layer, (3) stationarity and (4) uniform negative sampling, LightTune, when trained with stochastic gradient descent (SGD) for ReLU MLPs, satisfies*

$$\lim_{t\to\infty} \Pr(|\hat{y}^{(t)} - y_+^{(t)}| \geq \delta) = 0. \quad (10)$$

The proof will appear in the extended version of the paper.

Remark 3 (**On-Device Advantages of LightTune**). *A primary motivation for adopting the FF algorithm is its significant advantages for resource-constrained devices [13], [20], [21]: 1) **Dynamic Memory (RAM) Footprint.** Standard backpropagation requires storing all activations during the backward pass, hence the required RAM scales with the depth of the network and the layer sizes (i.e., $\sum_l M_l$ for MLP). In contrast, LightTune trains each layer locally which eliminates the necessity to store intermediate activations, meaning the peak memory footprint scales only with the size of the largest layer (i.e., $\max_l M_l$ for MLP). 2) **Implementation Simplicity.** Standard backpropagation necessitates a complex automatic differentiation (Autodiff) engine to build a computational graph and execute the chain rule. This typically requires a full deep-learning framework. LightTune avoids this overhead entirely.*

IV. Applications of LightTune

In this section, we leverage LightTune for short-term BLER prediction and link adaptation.

A. BLER Prediction

We consider the downlink short-term BLER prediction problem as an application of LightTune. Short-term BLER prediction at the UE-side is essential in cellular systems (e.g., 5G/6G) for enabling accurate CSI reporting. We begin by providing a brief background. The CSI reference signal (CSI-RS) period denotes the time interval between two consecutive reference signals used for CSI acquisition. This interval determines how frequently the receiver (i.e., UE) can update its channel estimates and report CSI to the transmitter (i.e., base station). CSI-RS transmissions can be configured as either periodic or aperiodic. In the periodic mode, reference signals are transmitted at regular intervals. In contrast, the aperiodic mode involves dynamically triggered transmissions, where the interval between the reference signals is variable.

The BLER is predicted at the beginning of the CSI-RS period by the ML model, using features such as the CSI-RS signal-to-noise ratio (SNR), and is denoted as $\hat{P}_{\text{BLER}}$, while the true BLER denoted as P_{BLER} is computed at the end as

$$P_{\text{BLER}} = \frac{n_{\text{NACK}}}{n_{\text{NACK}} + n_{\text{ACK}}}, \quad (11)$$

where n_{ACK} and n_{NACK} denote the number of PDSCH ACKs and NACKs in the CSI-RS period, respectively. Hence, the UE computes the BLER prediction error as

$$e(\hat{P}_{\text{BLER}}, P_{\text{BLER}}) = \left|\hat{P}_{\text{BLER}} - P_{\text{BLER}}\right|. \quad (12)$$

The fine-tuning process is triggered when the BLER prediction error reaches a pre-defined threshold δ. This condition ensures that model updates are performed only when the prediction deviates significantly from the empirical observation, enabling efficient and targeted fine-tuning.

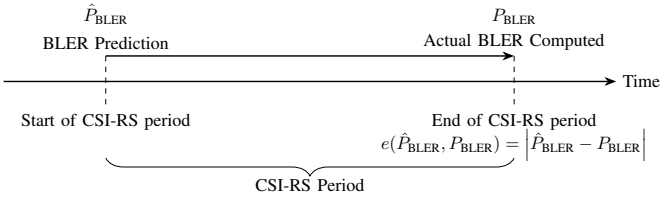


Fig. 3: Timeline showing BLER prediction at the start and actual BLER computation at the end of a CSI-RS period

Remark 4 (**Using the FF Algorithm for Regression**). *The FF algorithm was designed for classification tasks. We adapt it for regression through discretization, where the continuous variable to be estimated (i.e., BLER) is quantized into a finite set of discrete classes: $\{0, \lambda, 2\lambda, \dots\}$. This presents a trade-off: (1) prediction accuracy is bounded by the step λ; (2) using a finer λ increases the inference latency. As discussed in Sec. IV-B, our link adaptation application only requires a coarse estimate (i.e., if BLER is too high), justifying this approach.*

B. Channel Quality Indicator (CQI) Selection

In 5G, the UE reports a CQI to the base station, which reflects the perceived downlink channel conditions. Based on the reported CQI, the base station dynamically selects an appropriate modulation and coding scheme (MCS) to optimize throughput while maintaining reliable communication.

The online BLER prediction algorithm of Section IV-A can be integrated with a conventional table-based CQI selection algorithms such as the algorithm of [12] as a backoff mechanism. Traditional CQI selection methods rely on look-up tables that map the mean mutual information per bit (MMIB) or SNR to a CQI index. However, these methods may select a CQI index that results in an excessively high BLER, particularly initially when the algorithm has not yet converged.

To mitigate this, we propose an adaptive backoff strategy guided by online BLER prediction. Specifically, if the CQI selected by the conventional table-based algorithm yields a predicted

BLER greater than or equal to a predefined threshold τ_{BLER} (e.g., 0.9), the algorithm iteratively reduces the CQI index until the predicted BLER falls below the threshold. This reduction continues as long as the CQI does not drop below a minimum index CQI_{min} (e.g., 1). It is important to clarify that the loop in Algorithm 2 operates within a single CSI-RS reporting period. That is, the CQI is reported at the end of the loop.

Algorithm 2 Proposed CQI Backoff Algorithm: CQI-Tune

Input: CQI selected by table-based algorithm, and CQI_{min}
Output: Adjusted CQI with predicted BLER below τ_{BLER}
1: **while** $\hat{P}_{\text{BLER}}(\text{CQI}) \geq \tau_{\text{BLER}}$ **and** $\text{CQI} \geq \text{CQI}_{\text{min}}$ **do**
2: $\quad \text{CQI} \leftarrow \text{CQI} - 1$
3: $\quad$ Predict the BLER of this CQI, $\hat{P}_{\text{BLER}}(\text{CQI})$
4: **end while**
5: UE reports CQI to the base station

V. Simulation Results

We first experimentally validate our proposed loss function in Section V-A and then evaluate the performance of the proposed fine-tuning algorithm in Section V-B.

A. Experimental Validation of the Proposed Loss Function

To validate the proposed loss function, we compare the test accuracy with that of the Softplus loss function on MNIST dataset [22]. The settings of this experiment are summarized in Table I. Since the size of an MNIST image is 28×28, then the input layer in our MLP is of size 784 as the image is flattened first. In this experiment, positive and negative samples are constructed by replacing the first 10 pixels of an image by the one-hot encoding of the true label y_+ for positive samples and by the one-hot encoding of an incorrect label y_- for negative samples [8]. The details and code are available in [23].

Parameter	Value
Network Architecture	`[784, 500, 500]`
Activation Function	ReLU
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
Learning Rate α	0.03
Threshold T	2
Epochs	1,000

TABLE I: MNIST training configuration

Loss Function	Softplus	Proposed
Test Accuracy	93.15%	93.74%

TABLE II: Test accuracy on MNIST using Softplus loss vs. proposed loss

Our results in Table II demonstrate that the proposed loss function achieves comparable accuracy while offering enhanced computational efficiency.

B. BLER Prediction and CQI Selection

Our goal is to simulate scenarios in which the ML model is trained in an environment but evaluated in a different one to mimic the training-test mismatch problem. We use 12 features for the BLER prediction ML model, hence the input size is 13 since a label is attached to the feature vector.

Features. To predict the short-term BLER, we utilize features that capture both channel conditions and transmission parameters. These include the CSI-RS SNR and the CSI-RS capacity computed using the best precoding matrix indicator (PMI) for the reported rank indicator (RI), which helps characterize antenna correlation levels. We also incorporate the delay spread and Doppler frequency estimates to distinguish between different channel profiles. Additionally, we use the instantaneous PDSCH SNR along with the PDSCH SNR values from the three most recent transmissions, which aid in predicting the likelihood of NACKs. The feature set is completed by the current RI and CQI, the number of allocated resource blocks (RBs), and the number of demodulation reference signal (DMRS) symbols used in the transmission.

Training and Test Settings. We train and test our model on tapped delay line (TDL) channels [24], as summarized in Table III. In TDL channels, the letter (A, B, C) indicates the channel profile, while the number (30, 50, 200) specifies the root mean square delay spread (RMS DS) in nanoseconds. For both the training and the test settings, the bandwidth is 100 MHz, the number of RBs is 106, the sub-carrier spacing is 30 KHz and the CSI-RS period is 80 ms.

Parameter	Training	Test
Channels	TDL-A30	TDL-A30, TDL-B50, TDL-C200
SNR	Low (0–12 dB)	Low/Medium/High (0–40 dB)
Delay Profile	Low Delay	Low/High Delay
Doppler Frequency	Low (10 Hz)	Low/Medium (10–50 Hz)

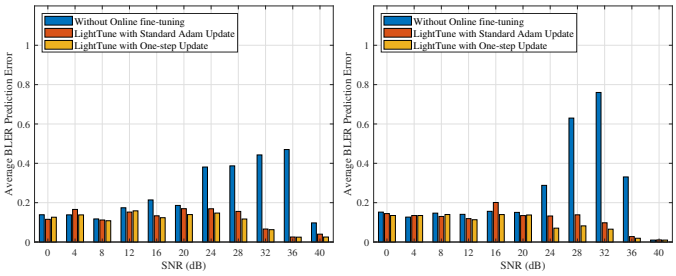
TABLE III: Training and testing configurations

Parameter	Value
Neural Network Size	`[13, 32, 32]`
Activation Function	ReLU
Offline Learning Rate α	0.03
Online Learning Rate α_f	0.03
Fine-tuning Threshold δ	0.3
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
Threshold T	9
Epochs	22,000
Training Samples	83,200
BLER Quantization Step	$\lambda = 0.1$

TABLE IV: Hyperparameters for the BLER prediction algorithm

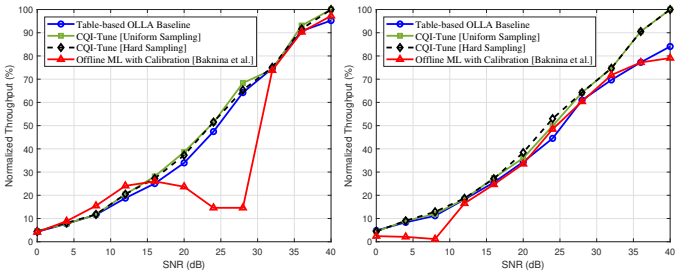
BLER Prediction error with and without Fine-tuning. We investigate which fine-tuning variant is better. Our results show that the one-step update approach is the best in terms of the BLER prediction error as shown in Fig. 4a and Fig. 4b.

- **BLER Prediction for TDL-A30 with SNR Mismatch.** Fig. 4a shows for SNR 0 to 12 dB that the BLER prediction error is almost the same with or without online fine-tuning since the baseline model is trained with TDL-A30 data from 0 to 12 dB. But above 12 dB, we see the significant decrease in the BLER prediction error due to enabling online fine-tuning. Specifically, the average BLER prediction error over all SNRs is reduced by 43.5%.
- **BLER Prediction for TDL-B50 with SNR and Channel Profile Mismatch.** Fig. 4b shows a more significant decrease in the BLER prediction error as a result of enabling online fine-tuning. This is because TDL-B50 has not been used in the training. Hence, the training-test mismatch leads to low performance of the baseline ML model. In particular, the average BLER prediction error decreased by 36%.



(a) TDL-A30 (Doppler freq. = 10 Hz) (b) TDL-B50 (Doppler freq. = 30 Hz)

Fig. 4: Average BLER prediction error with and without online fine-tuning with uniform sampling.



(a) TDL-B50 (Dop. freq. = 30 Hz). (b) TDL-C200 (Dop. freq. = 50 Hz)

Fig. 5: Throughput performance under different channels with $\tau_{\text{BLER}} = 0.9$, $\text{CQI}_{\text{min}} = \max(\text{CQI} - 1, 1)$, where CQI is the table-based selection.

Parameter	**Value**
Neural Network Size	$12 \times 64 \times 64 \times 1$
Training Learning Rate α	0.001 initially
Learning Rate Schedule	Decays over 10 steps, then restarts with cycles Peak learning rate remains constant Minimum rate is 1×10^{-5}
Activation Function	ReLU
Epochs	250
Training Samples	83,200

TABLE V: Hyperparameters of the offline ML-based approach in [16]

Throughput Results. Next, we compare the throughput performance of three approaches with CQI reporting based on: a table-based outer loop link adaptation (OLLA) method similar to the algorithm of [12], the offline ML-based method [16], and the table-based OLLA method augmented with the proposed backoff mechanism described in Algorithm 2.

- **CQI Selection for TDL-B50.** In this case, CQI-Tune with uniform and hard sampling yields average throughput gains of 6% and 5%, respectively, over the table-based baseline.
- **CQI Selection for TDL-C200.** In this case, CQI-Tune with uniform and hard sampling achieves throughput gains of 9.5% and 7.2%, respectively, over the table-based approach.

We note that the offline ML method [16] shows degraded performance due to a training-test mismatch, having been trained exclusively on TDL-A30 low SNR data. This occurs despite using ACKs/NACKs to calibrate the model output, highlighting the need for an online learning mechanism to help offline ML models generalize to unseen conditions.

VI. Conclusion

We introduced LightTune, an efficient online fine-tuning framework that enables post-deployment improvement of ML models solely using forward propagations. We demonstrated the effectiveness of LightTune in BLER prediction and link adaptation, showcasing its potential for real-time performance enhancement in wireless systems. While we have leveraged the FF algorithm for a simple UE-side implementation, the core principles of this opportunistic, buffer-less framework could be similarly applied to backpropagation-based frameworks.

References

[1] 3rd Generation Partnership Project (3GPP), "Study on Artificial Intelligence (AI)/Machine Learning (ML) for NR Air Interface," 3GPP, Technical Report 38.843, Release 18, 2023.

[2] P. Kaswan *et al.*, "Statistical AI/ML model monitoring for 5G/6G: Interference prediction case study," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2024.

[3] J. Xu *et al.*, "Learning to estimate: A real-time online learning framework for MIMO-OFDM channel estimation," *IEEE Transactions on Wireless Communications*, 2024.

[4] ——, "Learning at the speed of wireless: Online real-time learning for AI-enabled MIMO in NextG," *IEEE Communications Magazine*, 2024.

[5] Samsung, "AI/ML Use Cases and Framework for 6GR," 3GPP TSG RAN1 Meeting #122, Bengaluru, India, R1-2505588, Aug. 2025.

[6] V. Saxena, H. Tullberg, and J. Jaldén, "Reinforcement learning for efficient and tuning-free link adaptation," *IEEE Transactions on Wireless Communications*, vol. 21, no. 2, 2021.

[7] Q. An *et al.*, "DRAGON: A DRL-based MIMO Layer and MCS Adapter in Open RAN 5G Networks," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024.

[8] G. Hinton, "The Forward-Forward Algorithm: Some Preliminary Investigations," *arXiv preprint arXiv:2212.13345*, 2022.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3, 1992.

[10] T. Schaul *et al.*, "Prioritized Experience Replay," *ICLR*, 2016.

[11] D. Rolnick *et al.*, "Experience Replay for Continual Learning," *Advances in neural information processing systems*, vol. 32, 2019.

[12] E. Peralta *et al.*, "Outer loop link adaptation enhancements for ultra reliable low latency communications in 5G," in *IEEE 95th Vehicular Technology Conference:(VTC-Spring)*, 2022.

[13] M. O. Torres, M. Lange, and A. P. Raulf, "On Advancements of the Forward-Forward Algorithm," *arXiv preprint arXiv:2504.21662*, 2025.

[14] Z. Dong *et al.*, "Machine learning based link adaptation method for MIMO system," in *IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2018.

[15] T. Van Le and K. Lee, "Machine-learning-aided link-performance prediction for coded MIMO systems," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 3, 2021.

[16] A. Baknina and H. Kwon, "Adaptive CQI and RI Estimation for 5G NR: A Shallow Reinforcement Learning Approach," in *IEEE Global Communications Conference (GLOBECOM)*, 2020.

[17] Y. Huang, Y. T. Hou, and W. Lou, "DELUXE: A DL-based link adaptation for URLLC/eMBB multiplexing in 5G NR," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, 2021.

[18] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[20] B. Huang and A. Aminifar, "TinyFoA: Memory efficient forward-only algorithm for on-device learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[21] F. De Vita *et al.*, "μ-ff: on-device forward-forward training algorithm for microcontrollers," in *IEEE Conference on Smart Computing*, 2023.

[22] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, 1998.

[23] M. Pezeshki, "Implementation of Forward-Forward (FF) training algorithm," https://github.com/mpezeshki/pytorch_forward_forward, 2023.

[24] 3GPP, "Study on channel model for frequencies from 0.5 to 100 GHz," Tech. Rep. TR 38.901 V14.0.0, July 2017.