Machine Learning for Quantum Key Distribution Network Optimization

Student: Shek Lun Leung

Supervisor: Erik Svanberg, Giulio Foletto, Vaishali Adya

Examiner: Katia Gallo

Abstract

Quantum Key Distribution (QKD) enables secure communication through provably secure encryption keys based on laws of quantum mechanics, but practical implementations face challenges due to computational delays in traditional optimization methods and system imperfections, particularly on low-power devices like drones and Raspberry Pi. This study addresses these limitations by exploring neural networks as an efficient alternative to dual annealing for optimizing the secure key rate (R) in the decoy-state BB84 protocol. Building on Lim et al. (2014) for key rate validation and reproducing Wang and Lo (2019) for parameter prediction, we simulate R across fiber lengths (0–200 km) and post-processing block sizes ($n_X = 10^4$ to 10^9) using JAX, achieving secure distances of 135–180 km. Dual annealing, employing a two-phase global-local search, maximizes R but requires 45 minutes on 10 CPU cores, proving impractical for real-time applications. Conversely, a PyTorch-implemented neural network with a three-layer architecture (4-16-32-5 neurons, ReLU activation) trains in 5 minutes 30 seconds on a GPU, offering a 9-fold speedup and generalizing to unseen $n_X = 5 \times 10^8$ data with high accuracy (key rates from 10^{-2} to 10^{-7} bits/pulse). This efficiency supports dynamic QKD systems, such as satellite networks and the quantum Internet of Things, where rapid adaptation to atmospheric turbulence is critical. Leveraging hardware accelerators (e.g., Apple's Neural Engine), the approach enhances scalability and energy efficiency for low-power devices. Future work should enhance both: for dual annealing, GPU implementations (e.g., CUDA, OpenCL) could reduce bottlenecks despite its sequential nature; for neural networks, modeling detector imperfections (e.g., higher P_{dc}, after-pulsing, lower η_{Bob}), finite-key effects, variable channel losses (e.g., free-space links with turbulence), and optimized decoy parameters to counter PNS vulnerabilities could extend secure distances beyond 200 km. Larger datasets, higher n_X (e.g., 10^{10}), additional layers, or regularization may improve generalization and reduce biases, enabling the quantum Internet of Things and bridging simulation with practical deployment.

Contents

I. Introduction

Quantum Key Distribution (QKD) is a vital technology for enabling secure communication between two parties, ensuring that two parties can share encryption keys with provable security based on quantum mechanics. A key aspect of practical QKD systems is the establishment of tight security bounds, which are essential for ensuring robust security in real-world implementations. For instance, Charles CW Lim et al. (2014) provided concise and tight finite-key security bounds for QKD in the decoy state, offering a foundational framework for secure key rate calculations in practical settings [8]. Moreover, real-world QKD systems face limitations due to practical constraints like transmission time and system imperfections (e.g., detector efficiency). Parameter optimization, specifically choosing optimal signal intensities and their transmission probabilities, is crucial to maximizing the secure key rate. Traditional methods for this optimization, such as local search or brute-force global search, are computationally expensive and often require evaluating the key rate function for many parameter sets, leading to significant delays. In practice, these traditional methods become a bottleneck, especially in applications requiring real-time optimization. This poses a significant challenge for low-power devices, such as mobile phones, drones, and single-board computers (e.g., Raspberry Pi), which are expected to play a crucial role in future QKD systems. Due to their limited computational resources and power constraints, real-time optimization using traditional CPU-based methods becomes infeasible. For instance, QKD in free-space scenarios, such as satellite-ground communication or drone-based systems, faces dynamic environmental conditions like atmospheric turbulence, requiring rapid parameter adjustments to maintain secure communication.

To address these challenges, there has been a growing interest in applying machine learning to improve QKD performance. For example, Lu et al. (2018) applied recurrent neural networks (RNNs) to continuous-variable (CV) QKDs for noise filtering, improving overall performance [10]. Liu et al. (2018) used machine learning algorithms to predict and compensate for the evolution of light intensity over time, enhancing stability and performance [9]. Wang and Lo (2019) proposed using machine learning to predict optimal parameters for QKD systems, significantly improving efficiency and security by optimizing key generation processes [18]. These developments highlight the potential of machine learning to accelerate and optimize various aspects of QKD, from parameter prediction to security analysis.

This work aims to reproduce existing research on utilizing neural networks for efficient parameter optimization in QKD protocols, focusing on maximizing the secure key rate. An in-depth analysis will be presented on Wang and Lo's (2019) work using neural networks for optimal parameter prediction [18]. Additionally, the secure key rate calculations will build upon the security bounds established by Charles CW Lim et al. (2014) [8]. This project addresses the need for efficient optimization methods that can operate in real-time, especially on low-power devices.

The optimization approach, architecture, training process, and performance evaluation of the proposed neural network model will be thoroughly discussed. Neural networks offer significant advantages by enabling real-time inference with orders of magnitude less computation time than traditional methods. Once trained, they can efficiently handle large parameter spaces and multiple QKD protocols with minimal additional computational cost. This capability is particularly valuable for modern low-power devices, which often feature specialized hardware accelerators for neural network inference (e.g., Apple's Neural Engine, Movidius's Neural Compute Stick). These accelerators enable neural networks to achieve real-time optimization even in environments with limited computational capacity.

Moreover, neural networks can be pre-trained on high-performance hardware (e.g., desktop GPUs) and deployed on low-power devices. This flexibility ensures that even platforms with limited software or hardware support can benefit from machine learning-driven optimization. By leveraging these benefits, neural networks make it feasible to deploy QKD systems in portable, mobile, or distributed environments, such as the "quantum Internet of Things," where thousands of low-power devices must communicate securely and optimize their parameters in real-time. Thus, the adoption of neural networks not only improves computational efficiency but also paves the way for the next generation of secure quantum networks.

II. Background on QKD

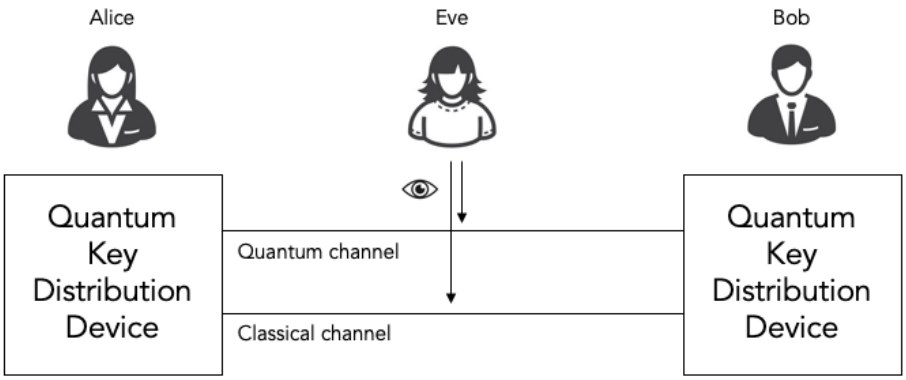

Figure 1: Illustration of a Quantum Key Distribution (QKD) system. Alice and Bob communicate securely using QKD devices over a quantum and classical channel. The quantum channel is used to transmit qubits, while the classical channel is employed for public communication. An eavesdropper, Eve, may attempt to intercept the quantum transmission, but any such attempt introduces detectable disturbances, ensuring security.

In 1984, Bennett and Brassard proposed the BB84 protocol for secure quantum key distribution (QKD) between Alice and Bob in an insecure environment [3]. Since then, the scheme has been widely studied and advanced both theoretically and practically. However, practical implementations of the BB84 protocol differ from the original proposal. Instead of using ideal single-photon sources, most systems employ weak pulsed laser sources, which can emit pulses containing more than one photon. This vulnerability allows an eavesdropper (Eve) to exploit the photon-number-splitting (PNS) attack, making high-loss channels particularly susceptible [14]. To counter the PNS attack, most BB84 implementations adopt the decoy-state method. In this approach, Alice randomly varies the mean photon number of each laser pulse sent to Bob, making it difficult for Eve to exploit photon-number-dependent attacks. By analyzing their shared data, Alice and Bob can detect photon-number-dependent loss in the channel. This method makes the BB84 protocol secure against PNS attacks, while also improving secret key rates and increasing tolerance to channel losses.

For an asymmetric coding BB84 protocol, the bases $\mathbf{X}$ and $\mathbf{Z}$ are chosen with probabilities P_X and $P_Z = 1 - P_X$, respectively. The secret key is extracted from the events that Alice and Bob both choose on the $\mathbf{X}$ basis. The protocol is based on the transmission of phase-randomized laser pulses and uses two-decoy settings. The intensity is randomly set to be one of the intensity μ_1, μ_2, μ_3 and satisfies the following condition for secure key generation. The intensity levels are subject to optimization, with μ_3 being fixed as a vacuum state. We consider a fiber-based QKD system model. Alice's laser pulses are set to one of the intensity levels, which are optimized within the constraints:

$$\mu_1 > \mu_2 + \mu_3 \tag{1}$$

$$\mu_2 > \mu_3 \geq 0 \tag{2}$$

Bob uses an active measurement setup with two superconducting nanowire single-photon detectors with a detection efficiency of $\eta_{\text{Bob}} = 10\%$, a dark count probability of $P_{\text{dc}} = 2.7 \times 10^{-7}$. The channel transmittance, η_{ch}, is calculated based on the fiber length L (km) and the attenuation coefficient, α.

$$\eta_{ch} = 10^{-\frac{\alpha L}{100}} \tag{3}$$

we assume $\alpha = 0.2$ dB/km. The fiber length is restricted to the range $L \in [0, 200]$.

The system transmittance is calculated with the detector efficiency, η_{Bob}, and channel transmittance, η_{ch}.

$$\eta_{sys} = \eta_{Bob}\eta_{ch} \tag{4}$$

Alice begins by randomly selecting a bit value, which is recorded as y_i. Then, she chooses a basis $a_i \in [\mathbf{X}, \mathbf{Z}]$ with probabilities P_X for the $\mathbf{X}$ basis and $P_Z = 1 - P_X$ for the $\mathbf{Z}$ basis. Additionally, she selects an intensity level $k_i \in \mathcal{K} = [\mu_1, \mu_2, \mu_3]$ with probabilities P_{μ_1}, P_{μ_2}, and $P_{\mu_3} = 1 - P_{\mu_1} - P_{\mu_2}$. Based on these choices, a laser pulse is prepared and sent to Bob via a quantum channel.

Bob selects a basis $b_i \in \{\mathbf{X}, \mathbf{Z}\}$ with probabilities P_X and $P_Z = 1 - P_X$, respectively. A measurement is then performed, and the outcome is recorded as y_i'. The measurement setup typically involves two single-photon detectors, resulting in four possible outcomes: $[0, 1, \emptyset, \perp]$. Here, 0 and 1 represent bit values, while $\emptyset$ denotes a no-detection event and $\perp$ represents a double-detection event. If the outcome is 0, 1, or $\emptyset$, the corresponding value is assigned to y_i'. If the outcome is $\perp$, a random bit value is assigned to y_i'.

During basis reconciliation, Alice and Bob publicly announce their basis and intensity choices over an authentic channel. They identify and verify the following sets for all $k \in \mathcal{K}$:

$$\mathcal{X}_k := \{i : a_i = b_i = \mathbf{X} \wedge k_i = k \wedge y_i' \neq \emptyset\}, \quad |\mathcal{X}_k| \geq n_{X,k} \tag{5}$$
$$\mathcal{Z}_k := \{i : a_i = b_i = \mathbf{Z} \wedge k_i = k \wedge y_i' \neq \emptyset\}, \quad |\mathcal{Z}_k| \geq n_{Z,k} \tag{6}$$

These steps are repeated until these conditions are met. The total number of laser pulses sent by Alice until these conditions are fulfilled is denoted as N.

First, a raw key pair (X_A, X_B) is generated by selecting a random sample of size n_X, where:

$$n_X = \sum_{k \in \mathcal{K}} n_{X,k}, \quad \mathcal{X} = \bigcup_{k \in \mathcal{K}} \mathcal{X}_k \tag{7}$$

Here, n_X is the post-processing block size. In this protocol, all intensity levels are utilized for key generation.

Second, the sets $\mathcal{Z}_k$ are announced, and the corresponding number of bit errors $m_{Z,k}$ is computed.

Third, the vacuum events ($s_{X,0}$, single-photon events ($s_{X,1}$) and phase errors ($c_{X,1}$ are calculated. Vacuum events ($s_{X,0}$)are instances where no photons are detected, corresponding to the intensity level μ_3, which is set close to zero to simulate a vacuum state. Single-photon events ($s_{X,1}$) occur when exactly one photon is detected, which is crucial for secure key generation as it minimizes the information available to an eavesdropper. Phase errors ($c_{X,1}$) refer to errors that occur in the phase of the single-photon states, which can compromise the security of the key if not properly accounted for.

If the required conditions for secure key generation, such as sufficient sample sizes and error rates, are not met, the protocol is aborted. Otherwise, post-processing proceeds.

Alice and Bob first perform an error-correction step that reveals at most λ_{EC} bits of information. This step is designed to correct for a predetermined error rate to ensure that both parties have identical keys. Following this, they conduct an error-verification step using a two-universal hash function. This step publishes a minimal amount of information to verify that their keys are identical, with a probability ϵ_{hash} that a pair of non-identical keys passes the verification. The information published during this step is typically very small and can be represented as $\log_2 \frac{1}{\epsilon_{\text{hash}}}$. Finally, conditioned on passing the error-verification step, Alice and Bob perform privacy amplification on their keys to extract a secret key pair (S_A, S_B), where $|S_A| = |S_B| = l$ bits. Conditioned on passing the checks in the error-estimation and error-verification steps, an ϵ_{sec}-secret key of length

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{cor}}}\right) \tag{8}$$

can be extracted, where the binary entropy function $h(x)$ is defined as:

$$h(x) = -x\log_2 x - (1-x)\log_2(1-x). \tag{9}$$

The number of vacuum events in $\mathbf{X}_A$ satisfies:

$$s_{X,0} \geq \tau_0 \frac{\mu_2 n^-_{X,\mu_3} - \mu_3 n^+_{X,\mu_2}}{\mu_2 - \mu_3}, \tag{10}$$

Where the Poisson probability that Alice prepares an n-photon state is:

$$\tau_n = \sum_{k\in\mathcal{K}} \frac{p_k e^{-k}\mu_k^n}{n!}. \tag{11}$$

The maximum and the minimum values of $n^{\pm}_{X,k}$ are:

$$n^{\pm}_{X,k} = \frac{e^k}{p_k}\left[n_{X,k} \pm \sqrt{\frac{n_X}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad n^{\pm}_{Z,k} = \frac{e^k}{p_k}\left[n_{Z,k} \pm \sqrt{\frac{n_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K}. \tag{12}$$

The number of single-photon events in $\mathbf{X}_A$ is:

$$s_{X,1} \geq \frac{\tau_1\mu_1\left[n^-_{X,\mu_2} - n^+_{X,\mu_3} - \frac{\mu_2^2-\mu_3^2}{\mu_1^2}\left(n^+_{X,\mu_1} - \frac{s_{X,0}}{\tau_0}\right)\right]}{\mu_1(\mu_2-\mu_3) - \mu_2^2 + \mu_3^2}. \tag{13}$$

Similarly, we calculate the number of vacuum events $s_{Z,0}$ and single-photon events $s_{Z,1}$ for $\mathcal{Z} = \bigcup_{k\in\mathcal{K}} \mathcal{Z}_k$ using Eqs. (10) and (13) with statistics from the Z basis. Additionally, the number of bit errors $\nu_{Z,1}$ associated with single-photon events in Z is required:

$$\nu_{Z,1} \leq \tau_1 \frac{m^+_{Z,\mu_2} - m^-_{Z,\mu_3}}{\mu_2 - \mu_3}, \tag{14}$$

Where:

$$m^{\pm}_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K}, \tag{15}$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}. \tag{16}$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [4]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right), \tag{17}$$

Where:

$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}. \tag{18}$$

The key rate becomes:

$$R = \frac{l}{N}, \tag{19}$$

where l is the length of the final secret key (in bits) and N is the total number of pulses sent. For more detailed derivation, please read the Appendix.

III. Background on Optimization

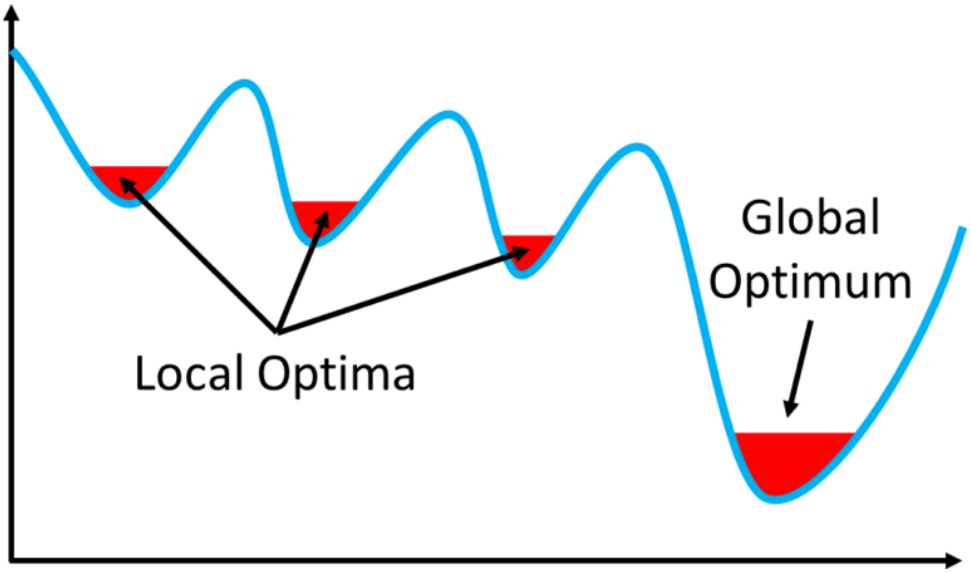

Figure 2: Local and global minima in optimization, showing how local search can converge prematurely. The given protocol has a non-convex key rate versus parameters function, such that the optimization problem is a convex optimization and a local search works. Extracted from [13].

Optimization in quantum key distribution (QKD) aims to maximize the secret key rate $R = l/N$ by tuning parameters like pulse intensity (μ_k) and its selection probability (P_{μ_k}). Local search algorithms (e.g., gradient descent) refine initial guesses incrementally, excelling in convex landscapes but struggling with non-convexity [1] (Fig. 2). Global search methods (e.g., simulated annealing) explore the full parameter space, mitigating this at a higher cost. Table 1 compares their traits.

Feature	**Local Search**	**Global Search**
Search Scope	Local region	Entire parameter space
Efficiency	Fast, efficient	Slow, resource-intensive
Risk of Local Minima	High	Low
Applicability	Convex functions	Non-convex landscapes

Table 1: Local vs. Global Search Algorithms

Dual Annealing extends simulated annealing by pairing probabilistic exploration with local refinement. It uses a temperature schedule $T(t)$ to accept suboptimal solutions early, avoiding local minima, then refines via Nelder-Mead as T decreases. Nelder-Mead, a derivative-free method, minimizes functions [11], suiting QKD optimization.

To address QKD's non-convex landscape, we employ Dual Annealing to globally search parameters (e.g., μ_k, P_{μ_k}), identifying regions maximizing R, followed by Nelder-Mead for precision. This hybrid approach outperforms standalone simulated annealing.

[1] A non-convex function has multiple local minima (as shown in the figure with "Local Minima" and "Global Optimum") This means the optimization landscape is complex, with several "valleys" (minima) where a local search might get stuck, potentially missing the global optimum.

IV. Background on Neural Networks

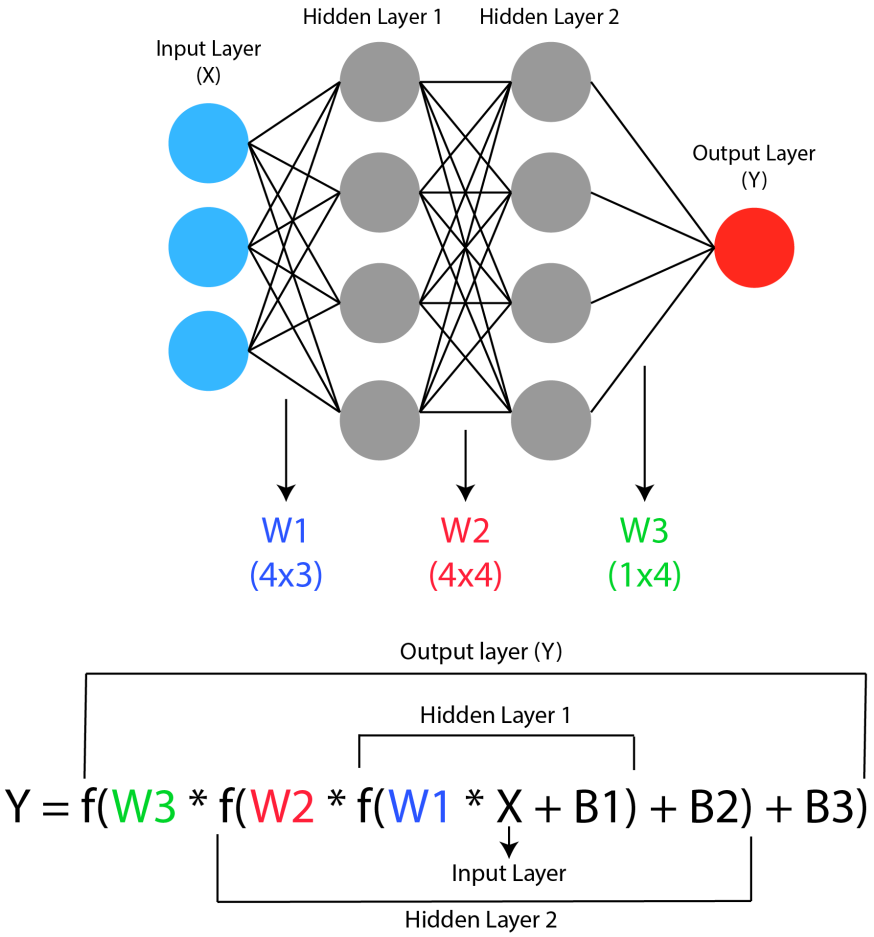

Figure 3: Artificial neural network with weighted inputs and activation function.

Neural networks (NNs), inspired by biological neurons, are powerful tools for optimizing complex problems, such as tuning quantum key distribution (QKD) parameters (e.g., intensities μ_k, probabilities P_{μ_k}) to maximize the secret key rate $R = \frac{l}{N}$. An NN consists of interconnected units called neurons, organized into layers: an input layer that receives features, hidden layers that perform computations, and an output layer that produces predictions. Each neuron computes an output as $\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$, where x_i are inputs (e.g., QKD parameters like L_{BC}, Y_0), w_i are weights, w_0 is a bias, and $g(\cdot)$ is a non-linear activation function, such as ReLU ($f(x) = \max(0, x)$) [5]. Data flows through the layers via weighted connections, with biases added at each step to adjust the computations. For example, Figure **??** illustrates a feedforward neural network with an input layer (X) of 3 nodes, two hidden layers (Hidden Layer 1 and Hidden Layer 2) with 4 nodes each, and an output layer (Y) with 1 node. The network transforms inputs through weight matrices W1 (4x3), W2 (4x4), and W3 (1x4), with bias terms B1, B2, and B3 applied at each layer. An activation function f (here simplified as a linear function $f(x) = x$ for clarity, though non-linear functions like ReLU or sigmoid are more common) is applied at each step. This architecture can approximate the complex loss landscapes of QKD, enabling efficient optimization of parameters to enhance the secret key rate.

A. Training Process and Propagation

The training of a neural network involves forward propagation and backward propagation, as illustrated in Figures **??** and 5. During forward propagation, input data is passed through the network, with each

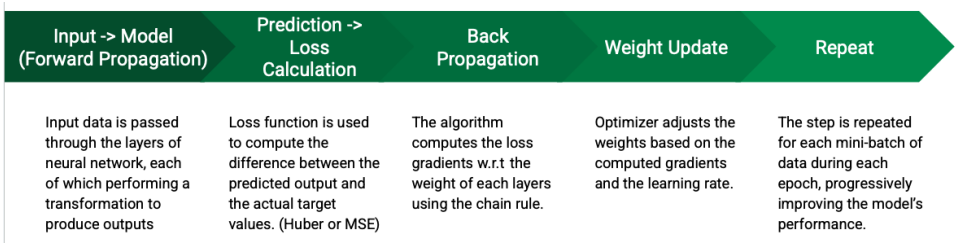

Figure 4: Forward and backward propagation in NN training.

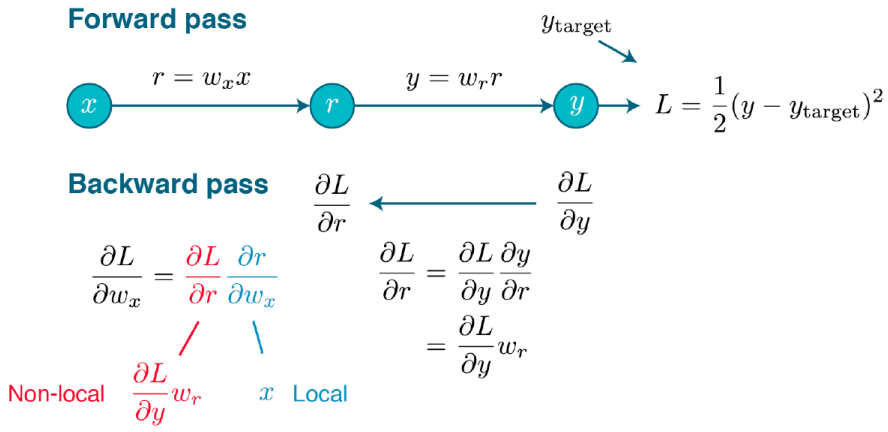

Figure 5: The details calculation of forward pass and backward pass.

neuron computing the weighted sum of its inputs followed by an activation function to produce outputs. The output from each layer serves as input to the next layer until the final output is obtained. This predicted output is compared to the actual target values using a loss function, such as Mean Squared Error (MSE), which quantifies the difference between predicted and actual values. In backward propagation, the loss is propagated back through the network, and gradients of the loss function concerning each weight are computed using the chain rule of differentiation. An optimization algorithm, such as Adam, then updates the weights and biases based on the computed gradients and the learning rate. This process repeats over multiple epochs, gradually minimizing the error and improving the network's accuracy.

B. Application to QKD

In this work, a fully connected NN with three hidden layers (16, 32, 16 neurons, ReLU) takes inputs like distance (L_{BC}), dark count (Y_0), misalignment (e_d), and signals (N), outputting five decoy-state parameters. Trained with Adam, it adapts to QKD's dynamic conditions, outperforming traditional optimization by rapidly predicting near-optimal R values in simulations.

V. Method

A. Simulation of Key Rate

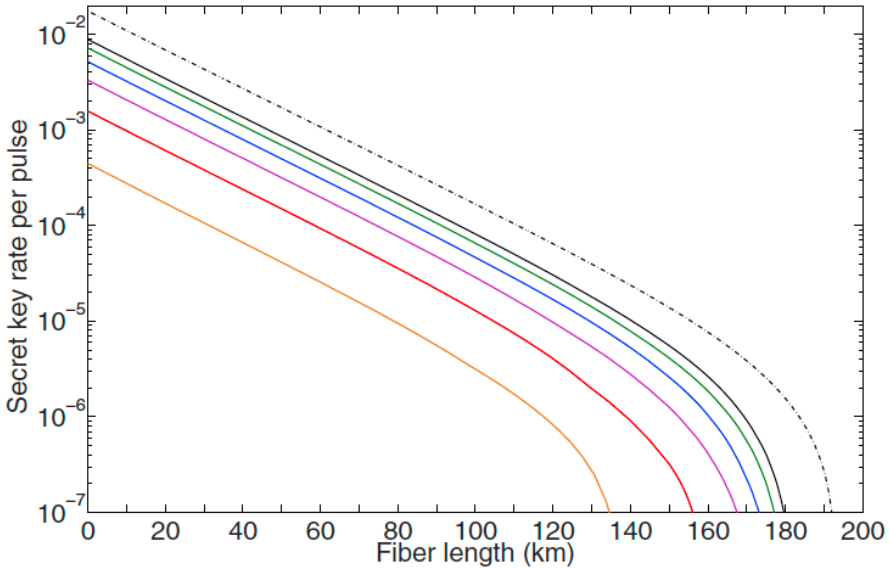

Figure 6: Secret key rate vs. fiber length for $n_X = 10^4$ to 10^9, from [8]. The secret key rate (color solid lines) is numerically optimized (in logarithmic scale) for fixed postprocessing block size $n_X = 10^s$ with $s = 4, 5, ..., 9$ (from left to right). The asymptotic secret key rate (dashed line), corresponds to the asymptotic secret key rate, i.e. in the limit of infinitely long keys. The number of laser pulses sent by Alice can be approximated with the secret key rate and the block size, i.e., $N \leq \frac{n_X}{R}$.

The QKD key rate was optimized using the BB84 decoy state protocol to establish a performance baseline before optimization, as depicted by [8]. This foundational step computes the secret key rate $R = l/N$ (Eq. 19 from Section "Background on QKD") across fiber lengths $L = 0$ to 200 km with 1000 discrete points and detected events $n_X = 10^4$ to 10^9, spanning six logarithmic scales. The number of laser pulses sent by Alice can be approximated with the secret key rate and the block size, i.e., $N \leq \frac{n_X}{R}$. Implemented in JAX (Python 3.9.20), the simulation leverages efficient array operations and potential automatic differentiation, enabling rapid computation of R under realistic conditions. The parameter n_X represents detected events in the X basis, critical for estimating error rates and single-photon yields, a cornerstone of decoy-state security analysis. Key parameters were sourced from [8], with security ensured by secrecy error $\epsilon_{sec} = 10^{-10}$ and correction error $\epsilon_{cor} = 10^{-15}$, adhering to sufficiently secure QKD standards. Detailed specifications are provided in Appendix **??**.

Core calculations were encapsulated in a modular script, `QKD_functions.py`, enhancing code reusability and maintainability. The simulation employed a grid search, iterating over 1000 L values and six n_X scales to cover practical scenarios, representing short-range laboratory tests to long-distance operational networks. Detection rates were modeled as:

$$D_k = 1 - (1 - 2P_{dc})e^{-\eta_{sys}\mu_k},$$

using fixed intensities $\mu_k = [0.6, 0.2, 2 \times 10^{-4}]$ and probabilities $P_{\mu_k} = [0.05, 0.6, 0.35]$. These intensities were selected to balance signal strength for key generation with a low value to detect multi-photon eavesdropping attempts, a strategy validated in decoy-state literature. Basis probabilities were set at $P_X = 0.5$ and $P_Z = 0.5$, providing an initial symmetric configuration for key extraction and error checking. Error correction was calculated via:

$$\lambda_{EC} = n_X f_{EC} h(e_{obs}),$$

where $f_{EC} = 1.16$ reflects typical reconciliation inefficiency, and $e_{obs} = \frac{m_x}{n_X}$ is the observed error rate, with $h(e_{obs})$ as the binary entropy function measuring information loss. This term quantifies bits sacrificed to correct errors, directly impacting the net key length l.

B. Optimization

The optimization framework determines the optimal secret key rate and parameters for the decoy-state BB84 protocol using a Jupyter Notebook. It involves defining system parameters, formulating an objective function, and employing a two-phase optimization strategy with global and local search methods.

The variables to be optimized are a set of 5 parameters, $[\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X]$, where μ_1, μ_2 are the signal and decoy intensities, and P_{μ_1}, P_{μ_2}, P_X are the probabilities of sending them and the basis choice probability. We unite these 5 parameters into one parameter vector $\vec{p}$.

Key parameters include photon intensities μ_1, μ_2 (with $\mu_3 = 2\times10^{-4}$ fixed to simulate the effect of sending an imperfect vacuum state) and probabilities $P_{\mu_1}, P_{\mu_2}, P_{\mu_3}$ (with $P_{\mu_3} = 1-P_{\mu_1}-P_{\mu_2}$), constrained within bounds $\mu_k \in [0,1]$ and $P_{\mu_k} \in [0,1]$ to maintain realistic photon emissions and probability sums. The basis probability P_X is optimized, with $P_Z = 1 - P_X$, adjusting the trade-off between key extraction in the X basis and error estimation in the Z basis. Fixed parameters include $\alpha = 0.2$ dB/km (fiber attenuation), $\eta_{Bob} = 0.1$ (detector efficiency), $P_{dc} = 6 \times 10^{-7}$ (dark count probability), and $f_{EC} = 1.16$ (error correction efficiency), reflecting typical experimental conditions validated in prior QKD studies.

The calculation of the secret key rate depends not only on the optimizable parameters $\vec{p}$ but also on the experimental parameters $\vec{e}$ that represent the physical conditions of the QKD system. These experimental parameters are transformed into a normalized form to enhance optimization stability and efficiency:

$$e_1 = \frac{L}{100}, \tag{20}$$

$$e_2 = -\log_{10} P_{dc}, \tag{21}$$

$$e_3 = e_d \times 100, \tag{22}$$

$$e_4 = \log_{10} N, \tag{23}$$

where L is the fiber length (km), P_{dc} is the dark count probability, e_d is the misalignment error rate, and N is the total number of pulses sent. These transformations normalize the magnitudes—e.g., scaling L from 0–200 km to 0.1–2 reduces its numerical dominance in the optimization process.

$$\text{Input} : \vec{e} = [e_1, e_2, e_3, e_4] = [L, P_{dc}, e_d, N] \tag{24}$$

$$\text{Output} : \vec{p} = [p_1, p_2, p_3, p_4, p_5] = [\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X] \tag{25}$$

This preprocessing facilitates the mapping from experimental conditions $\vec{e}$ to optimal protocol parameters $\vec{p}$, which can be conceptualized as:

$$\vec{e} \to \vec{p} \tag{26}$$

Therefore, the QKD secret key rate can be expressed as a function of both sets of parameters:

$$\mathrm{R} = \mathrm{R}(\vec{e}, \vec{p}) \tag{27}$$

Where R is the objective function dependent on the experimental parameters $\vec{e}$, which cannot be controlled by the users, and the user parameters $\vec{p}$, which can be adjusted to optimize performance.

This calculates the rate for a given fixed set of parameters and experimental parameters. To calculate the optimal rate, we need to calculate

$$\mathrm{R}_{\max}(\vec{e}) = \max_{\vec{p} \in P} \mathrm{R}(\vec{e}, \vec{p}) \tag{28}$$

Where P represents the feasible parameter space constrained by physical limitations. By maximizing R, we also determine the set of optimal parameters $\vec{p}_{\text{opt}}$. Note that $\vec{p}$opt is a function of $\vec{e}$ only, and the key objective in QKD optimization is to find these optimal parameters based on the given experimental conditions:

$$\vec{p}_{\text{opt}} = \arg\max_{\vec{p}\in P} \mathrm{R}(\vec{e}, \vec{p}) \tag{29}$$

The objective function, implemented in JAX, maximizes the secret key rate R via the `objective` function, including factors like channel losses (dependent on L), detection rates (influenced by η_{Bob}), and errors such as dark counts and misalignment. JAX's automatic differentiation provides insights into parameter impacts on R, while Just-In-Time (JIT) compilation optimizes repeated evaluations for speed.

The optimization process for QKD parameters proceeds in two distinct phases to maximize the secret key rate $R = \frac{l}{N}$. In the first phase, a dual annealing algorithm—a hybrid of simulated annealing and local search—performs a global search within the defined parameter bounds (e.g., $\mu_k \in [0, 1]$, $P_{\mu_k} \in [0, 1]$). This search starts from initial guess values, provided in Table 2 for different numbers of photon pulses, which were determined to be near-optimal based on multiple prior optimization runs. These initial guesses were obtained by iteratively optimizing parameters across increasing fiber lengths, using the optimized parameters from the previous length as the starting point for the next to avoid jumping between closely valued minima, thus providing effective starting points for the global search. The global search ensures that the algorithm explores beyond these starting points to identify potentially better solutions within the parameter space. In the second phase, a local optimization method refines the best solutions found during the global search, converging to the optimal values of μ_k and P_{μ_k} with higher precision.

Power / Parameters	μ_1	μ_2	P_{μ_1}	P_{μ_2}	P_X
10^4	0.65	0.15	0.05	0.61	0.425
10^5	0.62	0.24	0.10	0.70	0.55
10^6	0.68	0.30	0.14	0.74	0.66
10^7	0.55	0.34	0.15	0.75	0.75
10^8	0.54	0.375	0.16	0.775	0.83
10^9	0.52	0.40	0.18	0.785	0.88

Table 2: Initial guess values for different powers

The dual annealing algorithm employs a cooling schedule (1000 iterations, temperature range 1000 to 0.01) to transition from broad exploration to targeted refinement. At high temperatures, it explores widely to escape local optima; as the temperature decreases, an embedded local minimizer—typically a quasi-Newton method—refines each proposed solution, identifying high-R regions efficiently. This hybrid approach balances global coverage with local improvement, yielding a robust candidate solution.

Subsequently, the Nelder-Mead algorithm performs a dedicated local optimization, initializing from the dual annealing result and using simplex transformations (e.g., reflection, expansion, contraction) over a maximum of 500 iterations with a tolerance of 10^{-6}. This gradient-free method ensures high precision in the final parameter values, leveraging the geometric adaptability of the simplex to converge effectively on the optimal R.

During optimization, six datasets are generated for $n_X = 10^4$ to 10^9 (i.e., $10^4, 10^5, 10^6, 10^7, 10^8, 10^9$), each comprising 1000 data points, systematically exploring the parameter space across detection event scales from small experimental setups to large-scale implementations. An additional unseen dataset of 5×10^8 points is produced for neural network testing, evaluating generalization across intermediate n_X values not directly sampled, enhancing model robustness for real-world applications.

To boost efficiency, the *joblib* library parallelizes objective evaluations across four cores, significantly reducing runtime for the extensive parameter sets by distributing workload evenly. JAX's JIT compilation

targets `optimal_parameters` and `objective`, pre-compiling these functions into efficient machine code to minimize overhead during iterative calls, ensuring rapid convergence toward the optimal parameter set stored in `optimized_params`.

C. Neural Network

A neural network (NN) was developed to predict optimal parameters for the BB84 decoy-state quantum key distribution protocol (QKD), offering a data-driven alternative to traditional optimization techniques. Implemented in PyTorch within a Jupyter Notebook, the NN harnesses the Metal Performance Shaders (MPS) backend on an M2 Pro device to accelerate computations, leveraging GPU capabilities for efficient training. The architecture features an input layer with 4 neurons, accepting normalized experimental inputs that represent physical conditions such as fiber length, dark count probability, misalignment error rate, and total pulse count. Two fully connected hidden layers, with 16 and 32 neurons, respectively, use the Rectified Linear Unit (ReLU) activation function to introduce non-linearity, enabling the model to learn intricate patterns in the data. The output layer, with 5 neurons, predicts the optimal parameters $\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X$, which govern the intensity of the photons and the basis probabilities critical to maximizing the secret key rate R.

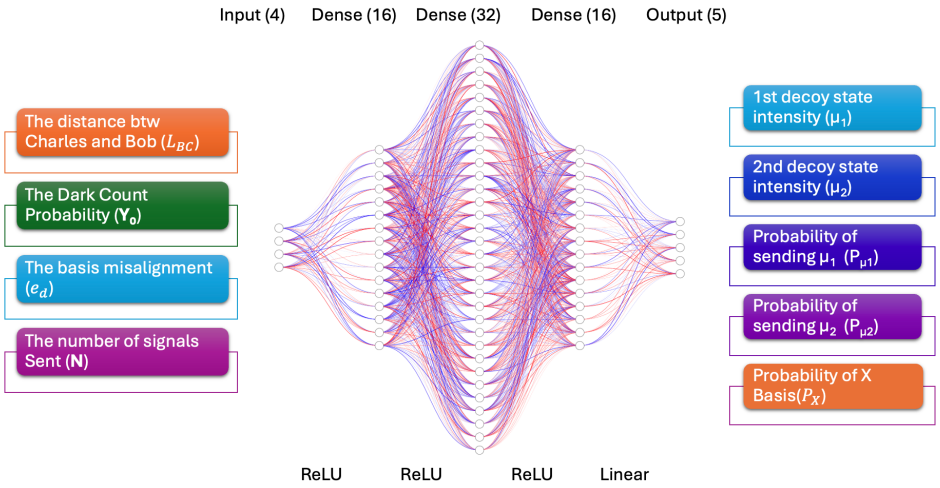

Figure 7: Neural network architecture for optimizing quantum key distribution (QKD) parameters. The network maps four input features to five output parameters through three dense layers with ReLU activation.

Figure 7 illustrates a neural network architecture designed to optimize the parameters of the quantum key distribution (QKD). The network takes four input features: the distance between Alice and Bob (L), the dark count probability (P_{dc}), the basis misalignment (e_d), and the number of signals sent (N). These inputs are processed through three dense hidden layers with 16, 32, and 16 neurons, respectively, using ReLU activation functions. The final output layer, employing a linear activation function, predicts five parameters critical to QKD: the first and second decoy state intensities (μ_1 and μ_2), the probabilities of sending these decoy states (P_{μ_1} and P_{μ_2}), and the probability of using the X basis (P_X).

The methodology for training and testing the neural network (NN) to optimize quantum key distribution (QKD) parameters is illustrated in Figure 8. The process begins with a generator program that produces $6\times1{,}000$ sets of random experimental parameters $\tilde{e}_{\text{popt}}$, each paired with their corresponding optimal parameters $\tilde{p}_{\text{opt}}$, derived through extensive simulations. These parameters encompass physical conditions such as fiber length, dark count probability, misalignment error rate, and total pulse count, with the optimal parameters including the decoy state intensities and basis probabilities. This dataset, totaling 6,000 points, is used to train the neural network via the NN trainer program, enabling the model to learn the mapping between experimental conditions and optimal QKD parameters. After training, the NN inference program predicts optimal parameters $\tilde{p}_{\text{pred}}$ from a single set of new experimental data $\tilde{e}_{\text{popt}}$, representing unseen scenarios. Finally, a validation program computes the secret key rates $R(\tilde{e}_{\text{popt}}, \tilde{p}_{\text{opt}})$

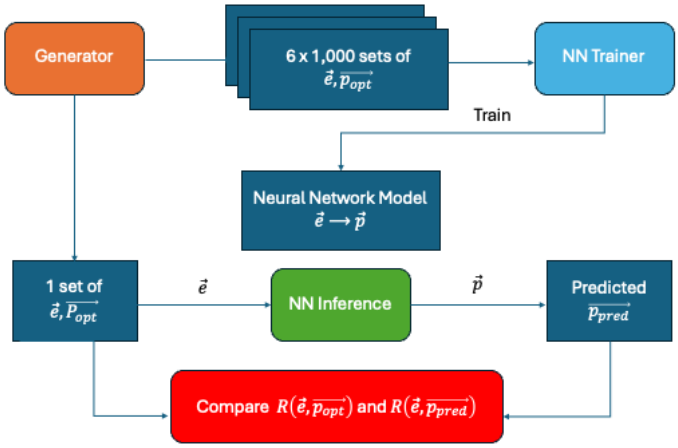

Figure 8: Data flow for the training and testing of the neural network (NN) in optimizing quantum key distribution (QKD) parameters. Rounded-corner boxes represent programs, and rectangular boxes represent data. The generator program produces 6×1,000 sets of random experimental parameters $\tilde{e}_{\text{popt}}$ and their corresponding optimal parameters $\tilde{p}_{\text{opt}}$, which are used to train the neural network. After training, the NN inference program predicts optimal parameters $\tilde{p}_{\text{pred}}$ from a single set of new experimental data $\tilde{e}_{\text{popt}}$. A validation program then compares the key rates $R(\tilde{e}_{\text{popt}}, \tilde{p}_{\text{opt}})$ and $R(\tilde{e}_{\text{pred}})$, evaluating the performance of the predicted parameters against the optimized benchmarks.

and $R(\tilde{e}_{\text{pred}})$, comparing the performance of the predicted parameters against the optimized benchmarks to assess the neural network's predictive accuracy.

The training dataset was constructed from previous optimization runs, covering a range of detection event scales from 10^4 to 10^9. Each data point pairs the normalized experimental parameters with the corresponding optimized parameters derived from extensive simulations. The experimental parameters were normalized, and the optimized parameters were scaled to a range [0, 1], preserving their relative magnitudes and facilitating stable training. The dataset, which totals 6,000 points (1,000 per scale), was filtered to include only scenarios with positive key rates and fiber lengths of up to 200 km, as the BB84 protocol with the chosen parameters (Table 3) results in a key rate that becomes zero or negative beyond this distance, reflecting practical QKD constraints. It was then divided into an 80% training set (4,800 points) and a 20% validation set (1,200 points), with random shuffling applied to eliminate bias and ensure representative sampling across conditions.

Training employed the Mean Squared Error (MSE) loss function to minimize the difference between predicted and target parameters. The Adam optimizer, starting with a learning rate of 10^{-2}, iteratively adjusts the model's weights over 5000 epochs, adapting to the data's complexity. A learning rate scheduler reduces the rate by a factor of 0.1 if validation loss stalls for 20 epochs, preventing overfitting and promoting convergence. Batches of 32 samples were processed using a data loader, leveraging the MPS device's parallel processing to expedite training. The model refines its predictions of $\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X$ through backpropagation, iteratively minimizing the loss and capturing the relationship between experimental conditions and optimal settings.

For evaluation, an unseen dataset with 5×10^8 detection events was generated, comprising 100 points across varying fiber lengths. Test inputs were normalized using the same scaling applied during training, ensuring consistency. The trained model predicted parameters, which were then inverse-transformed to their original scales and used to compute key rates via the analytical QKD model. These predicted rates were compared to optimized benchmarks, with results visualized as plots of key rates and parameters across fiber lengths (0–200 km) for the first and last training epochs. Loss curves for the training and validation phases were also plotted, demonstrating convergence and validating the model's performance. This approach efficiently predicts optimal QKD parameters, leveraging extensive data and normalization to generalize across scales, offering a robust tool for practical QKD applications.

VI. Results

A. Verification of BB84 key rate script

Results were visualized using Matplotlib, plotting R against L on a logarithmic scale (Fig. 9), which reveals R's exponential decay with distance due to increasing attenuation—a key challenge in QKD. This decay pattern aligns closely with trends reported in [8], confirming the simulation's fidelity and its utility as a benchmark. By mapping R's behavior across diverse conditions, this simulation provides essential insights into parameter feasibility, setting the stage for optimization to refine these values and enhance key rate performance.

The quantum key distribution (QKD) key rate was simulated using the decoy-state BB84 protocol to establish a performance baseline for optimization, evaluating the secret key rate R (bits per pulse) over fiber lengths $L \in [0, 200]$ km and detected events $n_X = 10^s$, with $s = 4, 5, \ldots, 9$ (i.e., 10^4 to 10^9). The decoy-state approach employs three intensity levels—signal and two decoy states—to bind eavesdropper information, enhancing security in realistic QKD systems. Functions from the "Background" section were implemented, with results benchmarked against Lim et al. (2014) [8].

Experimental Parameters $\vec{e}$	**Variable Parameters** $\vec{p}$
$\alpha = 0.2$	$\mu_1 = 0.6$
$\eta_{\text{Bob}} = 0.1$	$\mu_2 = 0.2$
$P_{\text{dc}} = 6 \times 10^{-7}$	$\mu_3 = 2 \times 10^{-4}$
$e_{\text{mis}} = 5 \times 10^{-3}$	$P_{\mu_1} = 0.05$
$f_{\text{EC}} = 1.16$	$P_{\mu_2} = 0.6$
$\epsilon_{\text{sec}} = 10^{-10}$	$P_{\mu_3} = 1 - P_{\mu_1} - P_{\mu_2} = 0.35$
$\epsilon_{\text{cor}} = 10^{-15}$	$P_X = 0.5$
$n = 1$	$P_Z = 1 - P_X = 0.5$

Table 3: Experimental Parameters and Variable Parameters for QKD Optimization

Figs. 9 and 6 illustrate the secret key rate R versus fiber length. Fig. 9, generated with the parameters ($P_{dc} = 6 \times 10^{-7}$, $P_{\mu_1} = 0.05$, $P_X = 0.5$), shows secret key rates (logarithmic scale) for $n_X = 10^4$ to 10^9. Fig. **??**, from [8], presents a similar analysis over a fiber optic link, including an asymptotic key rate (dashed) for infinite n_X, assuming three intensity levels and $N \leq n_X/R$. Both figures exhibit exponential decay in R with L due to channel losses ($e^{-\alpha L}$), halving every 15 km at $\alpha = 0.2\,\text{dB/km}$.

The simulation quantifies R gains via interpolation from Fig. 9. Larger block sizes reduce statistical fluctuations, improving R, but gains diminish beyond $n_X = 10^7$. The updated parameters lower R due to reduced signal contribution and increased noise.

The gains in secret key rate R diminish beyond $n_X = 10^7$, as statistical precision in parameter estimation stabilizes, consistent with the parameters ($P_{\mu_1} = 0.05$, $P_{\mu_2} = 0.6$, $P_{dc} = 6 \times 10^{-7}$). Under standard conditions ($e_{\text{mis}} = 5 \times 10^{-3}$), secure distances extend from approximately 120 km ($n_X = 10^4$) to 75 km ($n_X \geq 10^9$), with P_{dc} contributing to noise that limits performance at longer distances. Higher P_{μ_2} reduces R by increasing the proportion of weaker decoy states, lowering the signal contribution, while $P_X = 0.5$—reflecting equal X and Z basis usage in BB84—increases estimation overhead, further impacting R. Our decoy-state protocol accounts for multi-photon effects using three intensity levels, but limitations persist, such as neglecting after-pulsing in detectors, which may overestimate R in practice. These idealized assumptions influence the practical choice of n_X, where $n_X = 10^9$ balances R at 100 km) and computational cost for our simulated setup, supporting real-time QKD up to 175 km; however, this balance depends on desired key length, channel loss, and noise levels, suggesting a range of 10^6 to 10^9 may be optimal depending on the application. Future work could enhance the model by incorporating

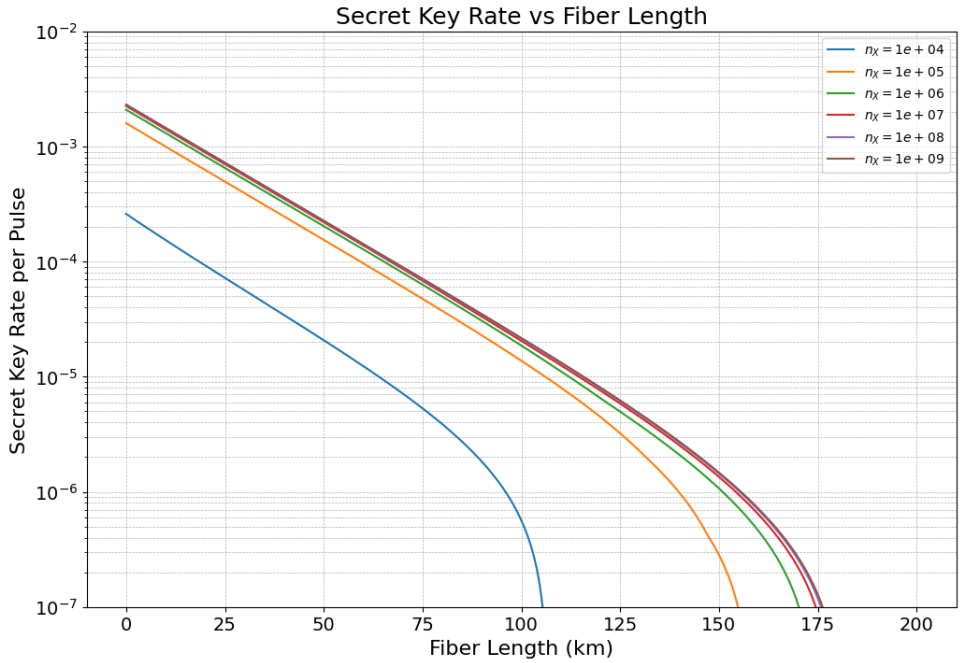

Figure 9: Secret key rate vs. fiber length, generated using functions from [8]. Numerically optimized secret key rates (logarithmic scale) are shown for post-processing block sizes $n_X = 10^s$, with $s = 4, 5, \ldots, 9$ (left to right). The trend aligns with Fig. 6.

detector imperfections (e.g., after-pulsing, increased P_{dc}) and finite-key effects, while leveraging improved detectors and higher n_X (e.g., 10^{10}) to extend secure distances beyond 200 km by reducing noise and improving statistical accuracy.

B. Optimization

The optimization results for $n_X = 10^4$ are presented in Figure 10. This figure consists of two panels showing the relationship between fiber length and key rate (left panel) and the optimized parameters (right panel).

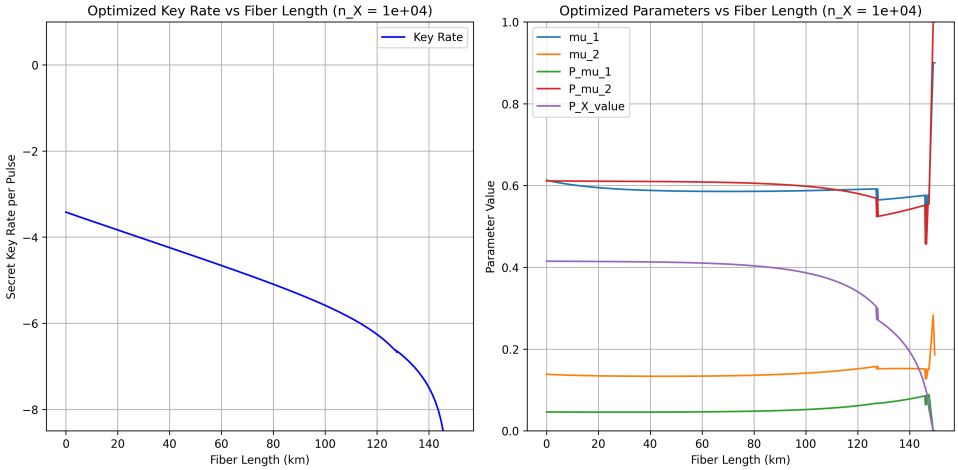

Figure 10: Left: Optimized key rate R (bits per pulse) versus fiber length L (km) for $n_X = 10^4$ pulses, showing exponential decay with a maximum secure distance of 145 km. Right: Optimized parameters (μ_1, μ_2, μ_3, P_{μ_1}, P_{μ_2}, P_X) versus fiber length, highlighting adaptive adjustments by the dual annealing algorithm, with notable fluctuations beyond 125 km due to extreme attenuation.

The left panel of Figure 10 displays the logarithmic (base 10) key rate as a function of fiber length. The key rate exhibits a monotonic decrease as the fiber length increases, with two distinct regions. From 0 to approximately 100 km, the key rate decreases at a relatively constant rate, following an approximately linear trend in the logarithmic scale. Beyond 100 km, the rate of decrease accelerates significantly, with the key rate dropping more rapidly as the fiber length approaches the maximum transmission distance of approximately 145 km. At short distances (0 km), the key rate is approximately $10^{-3.5}$ bits per pulse, while the distance decreases to approximately 10^{-8} bits per pulse. This behavior is consistent with the expected exponential attenuation of the signal in optical fibers, which becomes the dominant limiting factor at longer distances.

The right panel of Figure 10 illustrates how the optimal protocol parameters vary with fiber length. The signal intensity (μ_1, blue line) maintains a relatively stable value of approximately 0.6 for most of the transmission range up to about 125 km, after which it shows minor fluctuations. The decoy intensity (μ_2, orange line) remains consistently low (approximately 0.15) until about 125 km, beyond which it exhibits an increase and a notable spike near the maximum transmission distance. The probability for the signal state (P_{μ_1}, green line) begins at a low value (approximately 0.05) and gradually increases with distance, reaching its highest value at the maximum transmission distance. The probability for the decoy state (P_{μ_2}, red line) maintains a high value of approximately 0.6 for most distances, with a significant dip and then jump observed in the 125-145 km range. The X-basis probability (P_X, purple line) remains stable at approximately 0.4 until about 100 km, after which it decreases substantially as the fiber length approaches the maximum transmission distance.

The observed parameter variations reflect the protocol's strategic adaptations, as the optimizer maximizes the secret key rate R across diverse channel conditions while adhering to predefined security constraints ($\epsilon_{\text{sec}} = 10^{-10}$, $\epsilon_{\text{cor}} = 10^{-15}$). At shorter distances, where channel loss is minimal, parameters remain stable, indicating robust performance. Beyond 100 km, as channel loss increases, the protocol reduces the X-basis measurement probability P_X (from 0.5), reallocating measurements to the Z-basis to enhance error estimation, a trade-off that sacrifices some key generation speed to maintain security under the imposed constraints. Near the maximum transmission distance (140–145 km), pronounced fluctuations in parameters suggest the optimizer struggles to sustain a positive R under extreme attenuation, adjusting variables such as decoy intensities (μ_k) and probabilities (P_{μ_k}) as a final effort to extract secure key bits. These results highlight the protocol's adaptive capabilities, demonstrating how parameter optimization

maximizes R within security bounds, thereby influencing secure transmission distances, and underscoring its sensitivity to changing channel conditions to deliver a secure key.

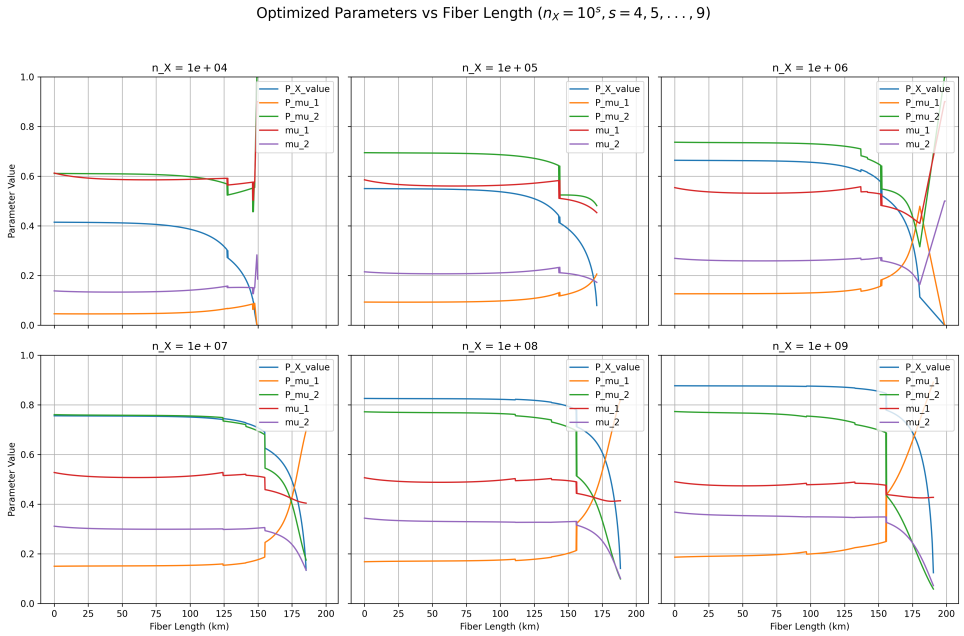

Figure 11: Optimized parameters (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X) versus fiber length (0–200 km) for $n_X = 10^4$ to 10^9, showing adaptive adjustments by the dual annealing algorithm. Parameters exhibit stability up to 100 km, with significant fluctuations (e.g., P_X drop, μ_2 spikes) beyond 125 km, reflecting responses to increasing attenuation.

Figure 11 provides a detailed view of how the parameters (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X) vary with fiber length across $n_X = 10^4$ to 10^9. At shorter distances, all parameters remain stable, reflecting robust operation. Beyond 100 km, P_X drops, reallocating measurements to the Z-basis for error estimation, while P_{μ_1} increases to prioritize signal states. The decoy intensity μ_2 spikes near maximum distances (145–180 km), probing multi-photon effects, and P_{μ_2} dips. These adjustments are more pronounced at lower n_X, with fluctuations decreasing and secure distances extending from 145 km ($n_X = 10^4$) to 180 km ($n_X = 10^9$) as statistical precision improves. This adaptability underscores the dual annealing algorithm's effectiveness in maximizing R.

Figure 12 illustrates the relationship between the key rate R and fiber length for $n_X = 10^4$ to 10^9, proportional to the number of signals in the QKD system. Plotted on a logarithmic scale ($\log_{10}$), R ranges from $\sim 10^{-3.5}$ to $\sim 10^{-2}$ at 0 km, decreasing to 0 at longer distances (145–175 km). As n_X increases, the key rate generally increases, expected due to a higher probability of successful key generation. However, the rate of increase diminishes at very high n_X values as the key rate approaches its asymptotic limit, determined by the channel's loss and the maximum achievable rate with infinite pulses. The key rate decreases with fiber length, an inevitable consequence of signal attenuation and increased noise, governed by a linear scaling with channel transmission efficiency $\eta = 10^{-\alpha L/10}$, where $\alpha = 0.2\,\text{dB/km}$, consistent with the decoy-state BB84 protocol. The figure represents the result of optimizing parameters (e.g., signal intensity, error correction, and detection efficiency) to maximize R for each fiber length and n_X value. The optimization process aims to balance trade-offs between key rate, distance, and signal count for optimal performance.

Figures 6, 9 and 12 collectively illustrate the secret key rate R as a function of fiber length in the decoy-state BB84 protocol, influenced by channel loss ($\alpha = 0.2\,\text{dB/km}$). Figure 9, representing unoptimized results, shows R decreasing from $\sim 10^{-3.5}$ (for $n_X = 10^4$) to $\sim 10^{-2.5}$ (for $n_X = 10^9$) at 0 km to $\sim 10^{-7}$ at 200 km across all n_X values, with a uniform maximum distance of 200 km. In contrast, Figure 12, optimized using dual annealing for $n_X = 10^4$ to 10^9, starts at $\sim 10^{-3.5}$ (for $n_X = 10^4$) to $\sim 10^{-2}$ (for $n_X = 10^9$) at 0 km, decreasing to $\sim 10^{-8}$ to $\sim 10^{-6}$ at distances of 145–175 km, respectively, reflecting significant performance gains. Figure **??**, optimized by [8], exhibits R from $\sim 10^{-2}$ at 0 km to $\sim 10^{-7}$ beyond 175 km while limited below 200 km, with multiple curves suggesting varied optimization

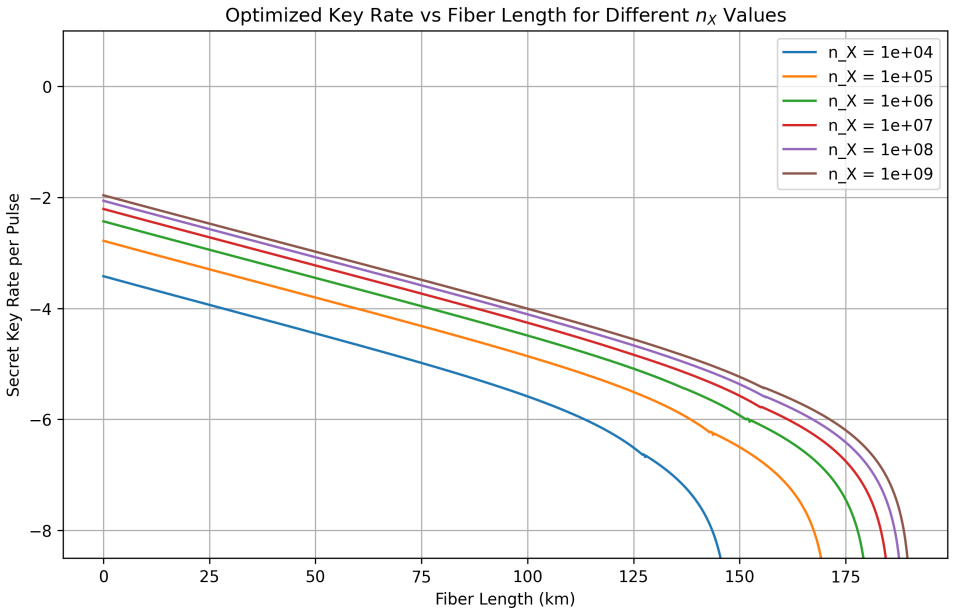

Figure 12: Optimzied Key Rate vs Fiber Length for Different n_X Values. The key rate is plotted on a logarithmic scale ($\log_{10}$) for various values of n_X, ranging from 10^4 to 10^9. The fiber length ranges from 0 to 175 km.

conditions. Compared to Figure 9, Figure 12 achieves up to an order of magnitude higher R (e.g., $\sim 10^{-2}$ vs. $\sim 10^{-3}$ at 0 km for $n_X = 10^9$) due to adaptive parameter adjustments (e.g., μ_2 spikes to ~ 0.6, P_X reductions to ~ 0.3, as seen in Figure 11), extending secure distances for higher n_X. Figure **??** outperforms Figure 9 similarly but covers a broader range (0–200 km). The explicit n_X variation in Figure 12 underscores its critical role in enhancing R, a factor likely fixed or less dynamic in Figures **??** and 9, highlighting the optimization's adaptability to signal count and distance.

C. Neural Network

The paper did not disclose much about the parameter values except for 50 km, $P_{dc} = 1.75 \times 10^{-7}$, $e_d = 0.0287$, $N = 2.99 \times 10^{12}$, $\alpha = 0.2$, $f_e = f_{EC} = 1.16$. We will use the values for the following parameters

$$\eta_{Bob} = 0.1 \qquad P_{ap} = 0.2$$
$$\epsilon_{sec} = 10^{-10} \qquad \epsilon_{cor} = 10^{-15}$$

Since the paper specified that $e_4 \in [11, 14]$, where $N \in [10^{11}, 10^{14}]$, the optimization will be run with the fixed post-processing block size $n_X = 10^s$ with $s = 6, 7, 8, 9$ for comparison with their result.

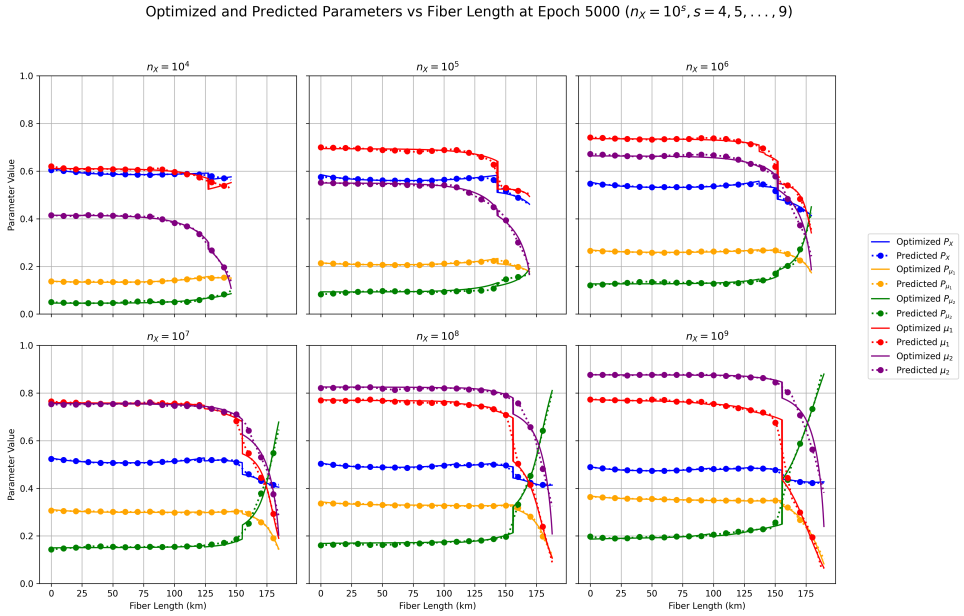

Figure 13: Predicted and Optimized Parameters vs. Fiber Length for Different n_X Values ($n_X = 10^s, s = 4, 5, \ldots, 9$). The subplots display the predicted and optimized values of parameters P_X, P_{μ_1}, P_{μ_2}, μ_1, and μ_2 as a function of fiber length (in km) for various n_X values, ranging from 10^4 to 10^9. Each subplot corresponds to a specific n_X, with fiber length on the x-axis (0 to 175 km, truncated where the key rate falls below 10^{-8}) and parameter values on the shared y-axis (0 to 1). A single shared legend on the right identifies the parameters, with solid lines for optimized values and dashed lines with circle markers for predicted values.

Figure 13 consists of a 2×3 grid of subplots, each illustrating the behavior of predicted and optimized parameters (P_X, P_{μ_1}, P_{μ_2}, μ_1, and μ_2) as a function of fiber length for different values of n_X. The n_X values range from 10^4 to 10^9, with each subplot corresponding to a specific n_X: the top row shows $n_X = 10^4$, 10^5, and 10^6, while the bottom row shows $n_X = 10^7$, 10^8, and 10^9. The x-axis represents the fiber length in kilometers (km), ranging from 0 to 175 km but truncated where the key rate falls below 10^{-8}, with the label "Fiber Length (km)" shown only on the bottom row to reduce redundancy. The y-axis represents the parameter value, ranging from 0 to 1, with the label "Parameter Value" shown only on the leftmost subplots (for $n_X = 10^4$ and 10^7) due to the shared y-axis. Each subplot is titled with the corresponding n_X value, formatted as $n_X = 10^s$, where s is the exponent (e.g., $n_X = 10^4$).

The parameters plotted are P_X (blue), P_{μ_1} (orange), P_{μ_2} (green), μ_1 (red), and μ_2 (purple). Solid lines represent optimized values, while dashed lines with circle markers represent predicted values, as indicated by a single shared legend on the right of the entire grid. The legend lists the parameters and their corresponding colors: Optimized P_X (blue solid line), Predicted P_X (blue dashed line with circles), Optimized P_{μ_1} (orange solid line), Predicted P_{μ_1} (orange dashed line with circles), Optimized P_{μ_2} (green solid line), Predicted P_{μ_2} (green dashed line with circles), Optimized μ_1 (red solid line), Predicted μ_1 (red dashed line with circles), Optimized μ_2 (purple solid line), and Predicted μ_2 (purple dashed line with circles).

Several trends are observable across the subplots. For smaller n_X values (e.g., 10^4, 10^5, 10^6), both predicted and optimized parameters remain relatively stable over longer fiber lengths, with the predicted values closely following the optimized values, indicating high predictive accuracy by the neural network after 5,000 epochs of iteration. For larger n_X values (e.g., 10^7, 10^8, 10^9), the parameters exhibit more subtle changes at longer fiber lengths, and the predicted values show small deviations from the optimized values, particularly at critical transition points around 150 km. The plots are truncated where the key rate falls below 10^{-8}, reflecting the increasing difficulty of maintaining a viable key rate over longer distances as n_X increases. The neural network performs well for smaller n_X, but its accuracy decreases for larger n_X, suggesting the need for further training or a more complex model to handle dynamic behavior at higher system scales.

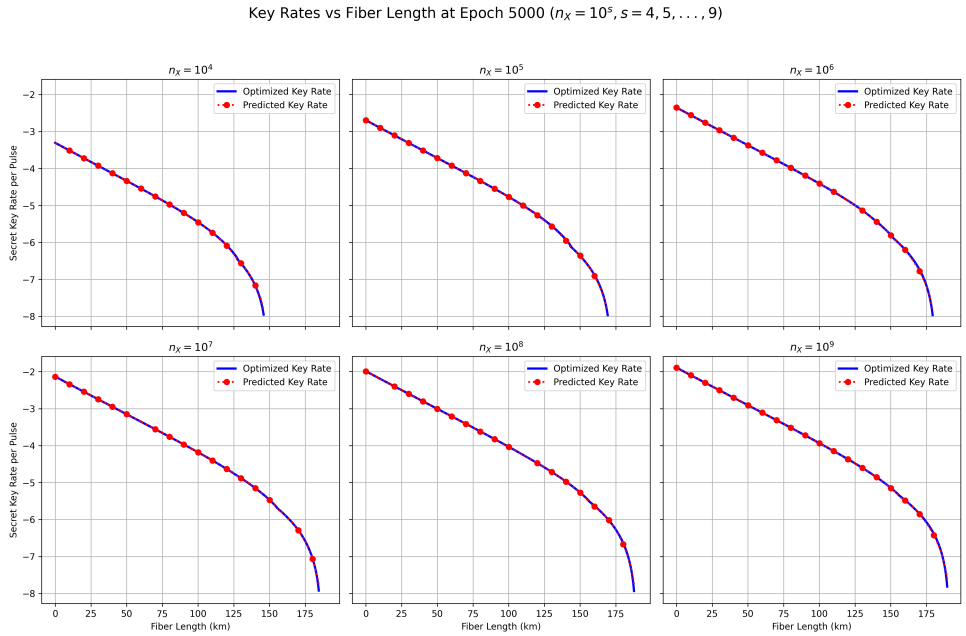

Figure 14: Predicted and Optimized Key Rates vs. Fiber Length for Different n_X Values ($n_X = 10^s, s = 4, 5, \ldots, 9$) at Epoch 5000. The subplots display the predicted and optimized secret key rates per pulse as a function of fiber length (in km) for various n_X values, ranging from 10^4 to 10^9. Each subplot corresponds to a specific n_X, with fiber length on the x-axis (0 to 175 km, truncated where the key rate falls below 10^{-8}) and the logarithmic key rate on the y-axis (-8 to -2). A single shared legend identifies the key rates, with solid blue lines for optimized values and dashed red lines with circle markers for predicted values.

Figure 14 presents a 2×3 grid of subplots illustrating the predicted and optimized secret key rates per pulse as a function of fiber length, evaluated across various n_X values (10^4 to 10^9) after 5,000 epochs of neural network training. Each subplot corresponds to a distinct n_X, arranged with $n_X = 10^4$, 10^5, and 10^6 in the top row, and 10^7, 10^8, and 10^9 in the bottom row. The x-axis denotes fiber length in kilometers (0 to 175 km), truncated where the key rate drops below 10^{-8}, with the label "Fiber Length (km)" appearing only beneath the bottom row to avoid repetition. The y-axis, scaled logarithmically from -8 to -2, represents the secret key rate per pulse and is labeled "Secret Key Rate per Pulse" solely on the leftmost subplots ($n_X = 10^4$ and 10^7) due to the shared axis. Subplot titles indicate the corresponding n_X value, formatted as $n_X = 10^s$ (e.g., $n_X = 10^5$). A unified legend on the right identifies the curves: solid blue lines for optimized key rates and dashed red lines with circle markers for predicted values, labeled "Optimized Key Rate" and "Predicted Key Rate", respectively.

Analysis of the subplots reveals no discernible differences between optimized and predicted key rates after 5,000 epochs. For all n_X values, both key rates exhibit a steady decline as fiber length increases, yet remain above 10^{-8} over extended distances—reaching approximately 150 km for $n_X = 10^4$ and up to 175 km for $n_X = 10^9$. The predicted key rates align precisely with their optimized counterparts across all subplots, showing no deviations and reflecting the neural network's robust predictive accuracy. This consistency holds for both smaller and larger n_X values, with the gradual reduction in key rates underscoring the challenge of sustaining secure communication over long distances.

Therefore, as the training process to the 5,000th epoch, the model coverages toward the optimized values are shown in Figure 14 and Figure 13. In Figure 13, the optimized parameters remain stable up to 100 km, reflecting the regime where channel loss is manageable, and then adjust beyond 100 km to compensate for increasing attenuation. The predicted parameters closely follow these trends, with dashed lines nearly overlapping solid lines, indicating that the neural network has accurately learned the parameter optimization landscape. In Figure Figure 14, the predicted key rate closely follows the optimized key rate, starting near 10^{-2} at 0km and decaying to approximately 10^{-7}, with minimal deviation across the range. This tight alignment of the predicted key rate to the optimized key rate indicates that the neural network has effectively learned this decay pattern. The close overlap of the red circles and blue line on the left panel suggests high predictive accuracy, with deviations so small they are nearly imperceptible across the 175 km range. It demonstrates the iterative training over 5,000 epochs has refined the model's weights to better match the dual annealing results with the predicted parameters showing smooth, controlled adjustments that mirror the optimized values. Minor deviations of the parameters at extreme distances (beyond 175 km) suggest slight residual biases, but the overall convergence is remarkable. Besides, the close match between predicted and optimized key rates for $n_X = 5 \times 10^8$ highlights the model's ability to generalize beyond its training data, which is a critical feature for practical applications where computational efficiency is needed for new scenarios. However, further training with a larger dataset or fine-tuning could address these residual biases, though their impact appears minimal given the overall performance.

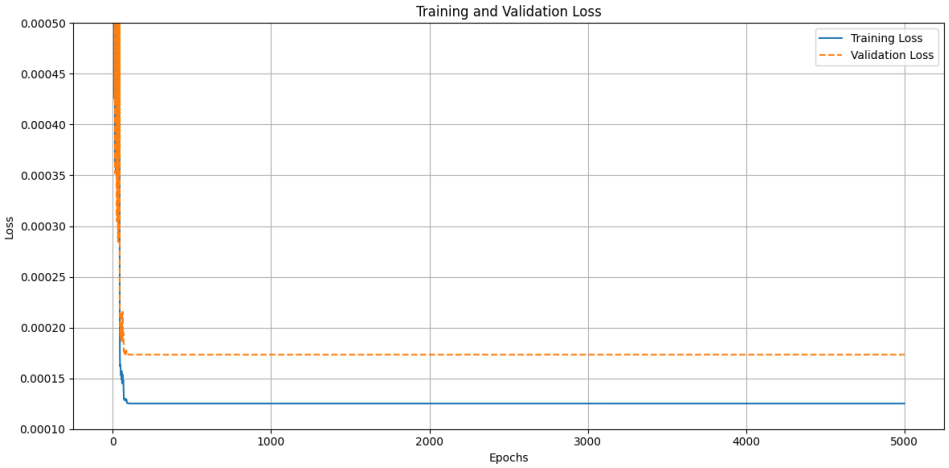

Figure 15: Training and Validation Loss vs. Epoches, showing convergence of the model with minimal difference between training and validation loss, indicating good generalization.

The training loss and validated loss are computed as averages over the respective datasets, weighted by the batch size, when are appended to lists and directly plotted in Figure 15. From around epoch 0 to epoch 500, both training and validation losses start at approximately 0.00050 and exhibit a steep decline. This rapid reduction indicates that the model quickly learns the basic patterns in the training data during the early stages of training. The similarity in the initial slopes of the training and validation loss curves suggests that the model generalizes well to the validation data at this stage, as both datasets benefit from the initial parameter updates driven by the optimizer.

After the initial drop, the training loss stabilizes around 0.00015 and the validation loss stabilizes around 0.00020, both plateauing by epoch 500. The plateau indicates that the model has largely converged, with further epochs yielding only marginal improvements. This rapid decrease within 500 epochs and subsequent stabilization align with the expected behavior of a well-tuned training process. The training loss consistently remains lower than the validation loss (0.00015 vs. 0.00020), as expected in neural network training, reflecting the model's optimization on the training data. The small gap between the two losses suggests that the model generalizes well to the validation data, with no significant divergence observed by epoch 500.

Overfitting would be evident if the training loss continued to decrease significantly while the validation

loss increased or plateaued at a higher value. In this plot, the validation loss does not rise after the initial drop, remaining stable and close to the training loss. This behavior suggests that the model generalizes reasonably well and does not overfit the training data excessively over the 500 epochs.

The training loop includes a learning rate scheduler, which adjusts the learning rate based on the validation loss. The stabilization of both loss curves after epoch 500 reflects the scheduler reducing the learning rate when the validation loss stops improving significantly, allowing the model to fine-tune its parameters without overshooting. The minor fluctuation in both losses after convergence suggests that the scheduler effectively reduces the learning rate after the initial learning phase, preventing large updates that could destabilize the converged model.

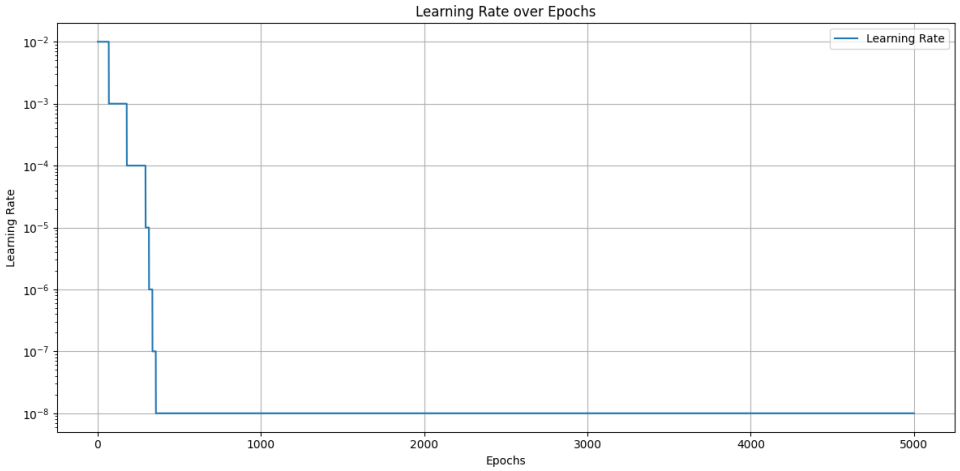

Figure 16: Learning Rate over epochs illustrates how the learning rate of a neural network model changes throughout its training process, spanning 5,000 epochs.

Figure 16 depicts a learning rate schedule that adjusts the learning rate dynamically. The learning rate determines the size of the steps the model takes when updating its parameters during optimization. The learning rate begins at a relatively high value of approximately 10^{-2} at epoch 0, which allows the model to make large updates to its parameters early in training. This accelerates learning by quickly reducing the loss and helping the model capture the general patterns in the data. It decreases in a stepwise manner over the initial epochs, dropping sharply from 10^{-2} to 10^{-3}, then to 10^{-4}, and continues this pattern until it reaches approximately 10^{-7}. These reductions are triggered by a plateau in the model's performance when the validation loss stops improving significantly. It allows the model to transition from coarse adjustments to finer adjustments as it approaches an optimal solution. After reaching 10^{-7} around epoch 1,000, the learning rate stabilizes and remains constant through to epoch 5,000. It indicates that the learning rate has hit a minimum value, where further reductions are unnecessary. At this stage, the model makes very small updates, focusing on fine-tuning its parameters to refine predictions without risking overshooting the optimal solution. This pattern is typical in neural networks, where step decay schedules help optimize performance.

Figure 17 presents a dual-panel plot evaluating the performance of a neural network in predicting parameters and key rates for the BB84 decoy-state quantum key distribution (QKD) protocol at $n_X = 5 \times 10^8$, after training for 5,000 epochs on a dataset of 6,000 points generated through optimization. An unseen dataset of 100 data points for $n_X = 5 \times 10^8$, with the 4 experimental parameters are used to predict the parameters and corresponding key rates. The left panel displays the logarithmic (base 10) secret key rate per pulse (R, in bits per pulse) as a function of fiber length (0 to 175 km), with the key rate ranging from -7 to -2. Optimized key rates, derived via dual annealing, are plotted as blue circles, while predicted key rates from the neural network are shown as red circles. The right panel illustrates the corresponding optimized and predicted parameter values (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X) over the same fiber length range, with parameter values ranging from 0 to 1. Optimized parameters are represented by solid lines, and predicted parameters by dashed lines, with distinct colors for each parameter: μ_1 (red), μ_2 (purple), P_{μ_1} (yellow), P_{μ_2} (green), and P_X (blue). A shared legend on the right identifies the line styles

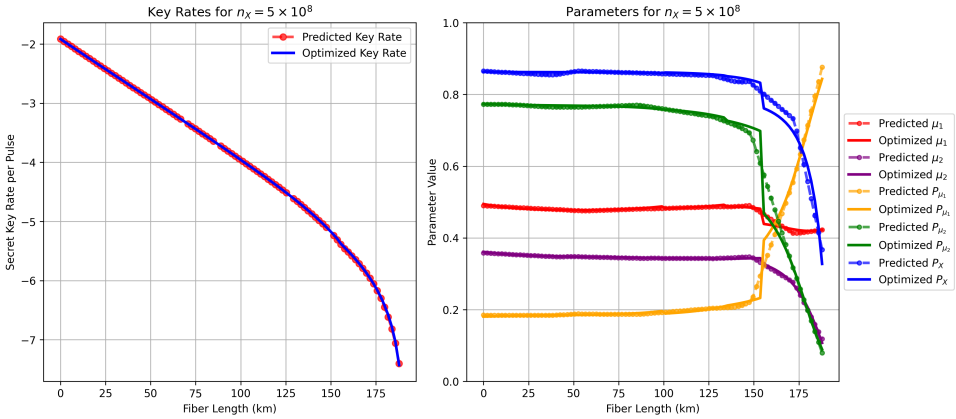

Figure 17: Comparison of Key Rates using neural network predicted parameters vs. using optimal parameters found by dual annealing for BB84 protocol, at different distances between Alice and Bob, for the last epoch of 5,000 iterations. Left: Logarithmic (base 10) secret key rate R (bits per pulse) versus fiber length (0–175 km) for $n_X = 5 \times 10^8$ with 100 data points, comparing optimized key rates (blue circles) with predicted key rates from a neural network (red circles), trained on a dataset of 6,000 data points (1,000 per $n_X = 10^s$ value, $s = 4, 5, \ldots, 9$). Right: Optimized (solid lines) and predicted (dashed lines) parameter values (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X) versus fiber length, illustrating the neural network's performance in approximating the dual annealing optimization results.

and colors for both optimized and predicted values.

Analysis of Figure 17 evaluates the neural network's performance in predicting QKD parameters and key rates for $n_X = 5 \times 10^8$. After 5,000 epochs of training on a dataset of 6,000 points generated through optimization, the neural network model is built, mapping experimental parameters to optimal QKD parameters. An unseen dataset of 100 data points for $n_X = 5 \times 10^8$, with experimental parameters including fiber length, dark count probability, misalignment error rate, and total pulse count, is then input into the model to predict the parameters (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X) and the corresponding key rates. In the left panel, both optimized and predicted key rates exhibit a gradual decline with increasing fiber length, dropping from approximately 10^{-2} at 0 km to below 10^{-7} at 175 km. The predicted key rates closely align with the optimized key rates across the entire range, with minimal deviations, indicating high predictive accuracy by the neural network. In the right panel, the optimized and predicted parameters demonstrate consistent behavior up to around 150 km, with P_X, P_{μ_1}, and P_{μ_2} remaining relatively stable (e.g., $P_X \approx 0.8$, $P_{\mu_1} \approx 0.4$, $P_{\mu_2} \approx 0.6$), while μ_1 and μ_2 are nearly constant at approximately 0.4 and 0.8, respectively. Beyond 150 km, the parameters undergo sharp transitions: P_X drops to near 0, P_{μ_1} and P_{μ_2} increase to around 0.8 and 0.9, and μ_1 and μ_2 rise to 0.8 and 1.0, respectively. The predicted parameters closely follow these trends, with slight deviations near the transition point (e.g., the predicted μ_2 reaches 1.0 slightly earlier than the optimized value). By comparing the predicted values with the optimized benchmarks, we observe that they are closely aligned, underscoring the neural network's ability to approximate the dual annealing optimization results effectively. However, the minor discrepancies at longer fiber lengths suggest potential for further refinement in capturing abrupt parameter changes.

Protocol	BB84
Device	MacBook Pro M2 Pro (Apple Silicon)
Dual Annealing for Optimization	using 10 CPU out of 12 cores (45 minutes for 1,000 data points)
Neural Network	GPU with 19 units (5 minutes 30 seconds for 5,000 epochs)

Table 4: Comparison of Computation Times for Optimization and Neural Network Training

There are practical reasons why neural networks may be preferred over dual annealing for optimization, particularly on ARM-based systems like the MacBook Pro M2 Pro (Apple Silicon), as shown in Table 4. From a hardware perspective, dual annealing relies exclusively on CPU resources, utilizing 10 out of 12 CPU cores and taking 45 minutes to process 1,000 data points. This high CPU usage can be problematic

for devices that need to perform concurrent tasks, such as control systems or real-time processing, as it leaves limited CPU capacity for other operations [16]. Moreover, dual annealing lacks support for GPU acceleration, not only on ARM-based GPUs like the 19-unit GPU in the MacBook Pro M2 Pro but also on other platforms, including Windows, Linux, and Intel GPUs. This limitation stems from the algorithm's sequential nature, which makes parallelization on GPUs inefficient without significant re-engineering [19, 6]. In contrast, neural network training can effectively utilize the GPU, completing 5,000 epochs in just 5 minutes 30 seconds, offloading the computational burden from the CPU and enabling better resource allocation for multi-tasking [5].

From a software perspective, dual annealing and key rate computation require a full optimization software stack, which can be challenging to deploy on ARM-based systems. For instance, SciPy's dual annealing implementation, a common choice for such tasks, is CPU-only and does not leverage GPU acceleration on any platform, including Windows, Linux, or Intel GPUs [15]. On ARM CPUs, such as the Apple Silicon in the MacBook Pro M2 Pro, some optimization libraries may require additional setup or emulation layers (e.g., Rosetta 2 on macOS), which can degrade performance [1]. Additionally, the absence of GPU support for dual annealing means that the optimization process cannot benefit from hardware acceleration, further limiting its efficiency on modern systems [12, 7]. Conversely, a pre-trained neural network, developed using data generated by linear solvers, can be deployed using lightweight inference frameworks (e.g., TensorFlow Lite), which are increasingly optimized for ARM architectures, including GPU support [17]. This reduces dependency on complex software installations and enhances portability across diverse hardware.

However, neural networks come with trade-offs. Training a neural network requires an initial computational investment and high-quality training data, and its accuracy depends on the model architecture and the representativeness of the training dataset [5]. While inference on ARM GPUs is efficient, deploying a pre-trained model may still require compatible runtime environments, though these are generally less resource-intensive than full optimization setups [2]. Additionally, dual annealing, despite its computational intensity and lack of GPU support, can provide more precise optimization results in certain scenarios, as it directly searches the solution space rather than relying on learned approximations [19]. Thus, the choice between dual annealing and neural networks should consider hardware constraints, software compatibility, and the specific requirements of the application, such as the need for real-time performance versus the need for exact optimization [16].

The neural network was trained on a dataset with n_X values ranging from 10^4 to 10^9, with the following number of data points per n_X: 736 for $n_X = 10^4$, 855 for $n_X = 10^5$, 902 for $n_X = 10^6$, 927 for $n_X = 10^7$, 942 for $n_X = 10^8$, and 948 for $n_X = 10^9$, totaling 5,310 data points. Training for 5,000 epochs took approximately 18 minutes to complete. After training, the model was tested on an unseen dataset with $n_X = 5 \times 10^8$, consisting of 100 data points, which took less than 1 second. The training time is worthwhile, as it results in an efficient neural network model that predicts the secret key rate in the BB84 protocol with high accuracy and the optimized parameters $(\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X)$ with reasonable accuracy, though deviations are observed at longer fiber lengths (beyond 150 km).

VII. Conclusion

The investigation into the decoy-state BB84 QKD protocol provides insights into secret key rate (R) optimization, comparing dual annealing and neural network approaches to address real-time optimization challenges on low-power devices. The simulation, based on Bennett and Brassard (1984) [3] with decoy-state and finite size effects [14], used JAX across 1000 fiber length points ($L = 0$ to 200 km) and six n_X scales (10^4 to 10^9). It confirmed an exponential decay of R with L, consistent with channel transmittance ($\eta_{ch} = 10^{-\alpha L/100}$, $\alpha = 0.2\,\text{dB/km}$) and detector efficiency ($\eta_{sys} = \eta_{Bob}\eta_{ch}$), aligning with Lim et al. (2014) [8]. Secure distances ranged from 135 km ($n_X = 10^4$) to 170–180 km ($n_X \geq 10^7$), with R gains diminishing beyond 10^7, supporting real-time QKD up to 175 km under idealized conditions.

Optimization via a two-phase approach—dual annealing (1000-iteration cooling schedule, temperature 1000 to 0.01) to explore the convex key rate landscape (Fig. **??**) [13], then Nelder-Mead (500 iterations, tolerance 10^{-6})—adjusted parameters (μ_1, μ_2, P_{μ_1}, P_{μ_2}, P_X). Using initial guesses from prior runs (Table 2) and parallelized with `joblib` on four cores, it extended secure distances from 145 km ($n_X = 10^4$) to 180 km ($n_X = 10^9$), with key rates dropping from $\sim 10^{-3.5}$ at 0 km to $\sim 10^{-8}$ at maximum distances. However, requiring 10 of 12 CPU cores for 45 minutes (Table 4) without GPU support, it is impractical for low-power devices like drones or Raspberry Pi, as noted by Tan et al. (2018) [16].

The neural network approach, implemented in PyTorch with an MPS backend, trained in 5 minutes 30 seconds over 5,000 epochs versus 45 minutes for 1,000 points (Table 4). With a three-layer architecture (4-16-32-5 neurons, ReLU activation), it used a 6000-point dataset (80/20 split), MSE loss, and Adam optimizer with a dynamic learning rate. After 5,000 epochs, it closely matched optimized key rates (from $\sim 10^{-2}$ at 0 km to $\sim 10^{-7}$ at 175 km for $n_X = 5 \times 10^8$, Fig. 17) via forward and backward propagation (Figs. **??**, 5) [5]. Leveraging GPU's 19 units and hardware accelerators like Apple's Neural Engine, it supports real-time optimization in dynamic settings (e.g., satellite-ground links), where turbulence or mobility losses demand rapid adjustments. Pre-training on high-performance hardware for deployment on low-power devices enhances efficiency and scalability for secure quantum networks on drones or single-board computers, though it requires initial training investment and high-quality data, with residual biases at longer distances needing refinement.

The choice between dual annealing and neural networks depends on hardware, software, and application—neural networks for efficiency in low-power, dynamic settings, and dual annealing for precision in controlled scenarios. This work aligns with Wang and Lo (2019) [18] and Lim et al. (2014) [8]. Future work should enhance both: for dual annealing, GPU implementations (e.g., CUDA, OpenCL) could reduce bottlenecks despite its sequential nature [19]; for neural networks, modeling detector imperfections (e.g., higher P_{dc}, after-pulsing, lower η_{Bob}), finite-key effects, variable channel losses (e.g., free-space links with turbulence), and optimized decoy parameters to counter PNS vulnerabilities [14] could extend secure distances beyond 200 km. Larger datasets, higher n_X (e.g., 10^{10}), additional layers, or regularization may improve generalization and reduce biases, enabling the quantum Internet of Things and bridging simulation with practical deployment.

A. Appendix

A. Details in Key Rate Equation

The secret key rate calculation is based on the work in [8]. The key rate R is defined as:

$$R = \frac{l}{N}, \tag{30}$$

where l is the length of the final secret key (in bits) and N is the total number of pulses sent.

Detection and Probability Models

The expected detection rate (excluding after-pulse contributions) for intensity level k is:

$$D_k = 1 - (1 - 2P_{dc})e^{-\eta_{sys}\mu_k}, \tag{31}$$

where $\eta_{sys} = \eta_{ch}\eta_{Bob}$ is the system transmittance, μ_k is the mean photon number for intensity level $k \in \{1, 2, 3\}$, and P_{dc} is the dark count probability.

The probability of detecting n photons given intensity μ_k follows a Poisson distribution:

$$P(n|\mu_k) = \frac{\mu_k^n e^{-\mu_k}}{n!}. \tag{32}$$

The joint probability of a detection event and intensity μ_k is:

$$P(\text{det}, \mu_k) = D_k P_{\mu_k}, \tag{33}$$

where P_{μ_k} is the probability of choosing intensity μ_k, satisfying $\sum_k P_{\mu_k} = 1$.

X-Basis Analysis

The probability of detection in the X basis for intensity μ_k is:

$$P(\text{det}, \mu_k, X) = P(\text{det}, \mu_k)P_X^2, \tag{34}$$

with the total detection probability in the X basis given by:

$$\sum_k P(\text{det}, \mu_k, X). \tag{35}$$

The conditional probability of intensity μ_k given a detection in the X basis is:

$$P(\mu_k|\text{det}, X) = \frac{P(\text{det}, \mu_k, X)}{\sum_k P(\text{det}, \mu_k, X)}. \tag{36}$$

The expected number of events for intensity μ_k in the X basis is:

$$n_{X,\mu_k} = n_X \times P(\mu_k|\text{det}, X), \tag{37}$$

where $n_X = \sum_k n_{X,\mu_k}$ is the total number of events in the X basis. The total number of pulses sent is estimated as:

$$N = \frac{n_X}{\sum_k P(\text{det}, \mu_k, X)}. \tag{38}$$

Z-Basis Analysis

Similarly, for the Z basis:

$$\begin{aligned} P(\text{det}, \mu_k, Z) &= P(\text{det}, \mu_k) P_Z^2, \\ \sum_k P(\text{det}, \mu_k, Z) &= \sum_k P(\text{det}, \mu_k, Z), \\ P(\mu_k|\text{det}, Z) &= \frac{P(\text{det}, \mu_k, Z)}{\sum_k P(\text{det}, \mu_k, Z)}, \\ n_{Z,\mu_k} &= n_Z \times P(\mu_k|\text{det}, Z), \\ n_Z &= \sum_k n_{Z,\mu_k}, \end{aligned} \tag{39}$$

with an alternative expression:

$$n_{Z,\mu_k} = N D_k P_{\mu_k} P_Z^2. \tag{40}$$

Finite-Key Security Bounds

To account for finite-key effects, security bounds for n_{X,μ_k} and n_{Z,μ_k} are:

$$n_{X,k}^{\pm} = \frac{e^k}{p_k} \left[n_{X,k} \pm \sqrt{\frac{n_X}{2} \ln \frac{21}{\epsilon_{\text{sec}}}} \right], \quad n_{Z,k}^{\pm} = \frac{e^k}{p_k} \left[n_{Z,k} \pm \sqrt{\frac{n_Z}{2} \ln \frac{21}{\epsilon_{\text{sec}}}} \right], \tag{41}$$

where ϵ_{sec} is the security parameter, and $k \in \mathcal{K}$.

The probability that Alice prepares an n-photon state is:

$$\tau_n = \sum_{k \in \mathcal{K}} \frac{p_k e^{-\mu_k} \mu_k^n}{n!}. \tag{42}$$

Decoy-State Analysis

Using decoy-state analysis, the number of vacuum and single-photon events in the X basis ($s_{X,0}$ and $s_{X,1}$) are bounded as:

$$s_{X,0} \geq \tau_0 \frac{\mu_2 n_{X,\mu_3}^- - \mu_3 n_{X,\mu_2}^+}{\mu_2 - \mu_3}, \tag{43}$$

$$s_{X,1} \geq \frac{\tau_1 \mu_1 \left[n_{X,\mu_2}^- - n_{X,\mu_3}^+ - \frac{\mu_2^2 - \mu_3^2}{\mu_1^2} (n_{X,\mu_1}^+ - \frac{s_{X,0}}{\tau_0}) \right]}{\mu_1(\mu_2 - \mu_3) - \mu_2^2 + \mu_3^2}. \tag{44}$$

Similar bounds apply to $s_{Z,0}$ and $s_{Z,1}$ in the Z basis.

Error Probability

For a dedicated fiber (basis-independent error probability), the bit error probability for intensity k is:

$$e_k = P_{dc} + e_{\text{mis}}[1 - e^{-\eta_{sys}\mu_k}] + P_{ap} \frac{D_k}{2}, \tag{45}$$

where P_{ap} is the after-pulse probability, and e_{mis} is the misalignment error probability.

The expected number of errors in the Z basis for intensity μ_k is:

$$m_{Z,k} = e_k N_Z P_{\mu_k}, \tag{46}$$

where $N_Z = NP_Z^2$, and the total number of errors in the Z basis is:

$$m_Z = \sum_{k \in \mathcal{K}} m_{Z,k}. \tag{47}$$

Security bounds on the number of errors are:

$$m_{Z,k}^{\pm} = \frac{e^{\mu_k}}{p_k} \left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2} \ln \frac{21}{\epsilon_{\text{sec}}}} \right]. \tag{48}$$

The number of bit errors for single-photon events in the Z basis is bounded by:

$$v_{Z,1} \leq \tau_1 \frac{m_{Z,\mu_2}^{+} - m_{Z,\mu_3}^{-}}{\mu_2 - \mu_3}. \tag{49}$$

Phase Error Rate

The phase error rate for single-photon events in the X basis is:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{v_{Z,1}}{s_{Z,1}} + \gamma \left(\epsilon_{\text{sec}}, \frac{v_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1} \right), \tag{50}$$

where $c_{X,1}$ is the number of phase errors, and the finite-key correction term is:

$$\gamma \left(\epsilon_{\text{sec}}, \frac{v_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1} \right) = \sqrt{\frac{(s_{Z,1} + s_{X,1})(1 - \frac{v_{Z,1}}{s_{Z,1}}) \frac{v_{Z,1}}{s_{Z,1}}}{s_{Z,1} s_{X,1} \ln 2} \log_2 \left(\frac{s_{Z,1} + s_{X,1}}{s_{Z,1} s_{X,1} (1 - \frac{v_{Z,1}}{s_{Z,1}}) \frac{v_{Z,1}}{s_{Z,1}}} \frac{21^2}{\epsilon_{\text{sec}}^2} \right)}. \tag{51}$$

Error Correction and Observed Error Rate

The observed error rate in the X basis is:

$$e_{\text{obs}} = \frac{m_X}{n_X}. \tag{52}$$

The error correction term is:

$$\lambda_{EC} = n_X f_{EC} h(e_{\text{obs}}), \tag{53}$$

where f_{EC} is the error correction efficiency, and $h(x) = -x \log_2 x - (1 - x) \log_2 (1 - x)$ is the binary entropy function.

Final Secret Key Length

The secret key length is:

$$l = s_{X,0} + s_{X,1} - s_{X,1} h(\phi_X) - \lambda_{EC} - 6 \log_2 \left(\frac{2}{\epsilon_{\text{sec}}} \right) - \log_2 \left(\frac{2}{\epsilon_{\text{cor}}} \right), \tag{54}$$

where ϵ_{cor} is the error correction security parameter. The terms $-6 \log_2 \left(\frac{2}{\epsilon_{\text{sec}}} \right)$ and $- \log_2 \left(\frac{2}{\epsilon_{\text{cor}}} \right)$ account for security overheads.

B. Details in Optimization

This appendix details the technical implementation of optimizing parameters for the BB84 Quantum Key Distribution (QKD) protocol with decoy states, aimed at maximizing the secret key rate over varying fiber lengths and detection event counts. The simulation and optimization follow the analytical model in [8].

C. Parameter Space

The optimization explores:

- **Fiber Lengths (L)**: 0 to 200 km, with 100 steps (2 km increments), yielding 100 unique values.
- **Detection Events (n_X)**: $[10^4, 10^5, 10^6, 10^7, 10^8, 10^9]$ (6 values), though some runs use subsets (e.g., $[10^{10}]$).
- **Combinations**: Up to $100 \times 6 = 600$ combinations of (L, n_X), varying by experiment.

Optimized parameters are $\vec{p} = [\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X]$:

- μ_1, μ_2: Mean photon numbers for two intensity levels.
- P_{μ_1}, P_{μ_2}: Probabilities of selecting μ_1 and μ_2.
- P_X: X-basis probability, with $P_Z = 1 - P_X$.
- Fixed: $\mu_3 = 2 \times 10^{-4}$, $P_{\mu_3} = 1 - P_{\mu_1} - P_{\mu_2}$.

Bounds: $\mu_1 \in [4 \times 10^{-4}, 0.9]$, $\mu_2 \in [2 \times 10^{-4}, 0.5]$, $P_{\mu_1}, P_{\mu_2}, P_X \in [10^{-12}, 1 - 10^{-12}]$.

Fixed system parameters are listed in Table 5:

Table 5: Simulation parameters for the decoy-state BB84 protocol.

Parameter	**Value**	**Description**
α	0.2 dB/km	Fiber attenuation coefficient
η_{Bob}	0.1	Detector efficiency at Bob's side
Y_0	6×10^{-7}	Dark count probability
e_d	5×10^{-3}	Misalignment error rate
P_{ap}	0	After-pulse probability
f_{EC}	1.16	Error correction efficiency
ϵ_{sec}	10^{-10}	Secrecy error tolerance
ϵ_{cor}	10^{-15}	Correlation error tolerance

D. Optimization Process

The secret key rate $R = l/N$ is maximized using a hybrid approach:

- **Global Search**: `dual_annealing` (SciPy) explores the parameter space, avoiding local optima. Initial guess varies by n_X (e.g., $[0.54, 0.375, 0.16, 0.775, 0.83]$ for $n_X = 10^8$).

- **Local Refinement**: `minimize` with Nelder-Mead refines the solution (`maxiter=10000`, `xatol=1e-10`, `fatol=1e-10`).

The objective function is:

$$\text{Objective} = -R(\vec{p}, L, n_X), \tag{55}$$

computed via custom functions in `QKD_Functions.py`, including channel transmittance (η_{ch}), system efficiency (η_{sys}), detection probabilities (D_k), counts (n_{X,μ_k}), errors (m_{X,μ_k}), bounds ($S_{X,0}$, $S_{X,1}$, $v_{Z,1}$), and final key rate.

Validation against [8] shows close agreement (e.g., $R = 1.23 \times 10^{-3}$ vs. 1.25×10^{-3} at $L = 0\,\text{km}$, $n_X = 10^9$).

E. Parallelization

Optimization is parallelized using `joblib.Parallel` and `concurrent.futures.ThreadPoolExecutor` (12 threads), processing batches (size 16) with progress tracked via `tqdm-joblib`.

F. Framework Selection: JAX

JAX is used for its efficiency in numerical optimization:

- **Features**: JIT compilation, automatic differentiation (`grad`), vectorization (`vmap`).
- **Configuration**: Double precision (`jax_enable_x64=True`).

G. Dataset Generation

Results are stored in a JSON dataset:

- **Features**: $e_1 = L/100$, $e_2 = -\log_{10}(Y_0)$, $e_3 = e_d \times 100$, $e_4 = \log_{10}(n_X)$, key rate R, optimized parameters $\vec{p}_{opt}$.
- **Format**: Grouped by n_X, sorted by L (e.g., `qkd_grouped_dataset_YYYYMMDD_HHMMSS.json`).
- **Size**: Varies (e.g., 600 entries for 100 L and 6 n_X).

The codebase is available at `https://gits-15.sys.kth.se/slleung/QKD_KeyRate_ParameterOptimization/blob/main/Analysis/BB84_Parameters_2014_Analysis_Jax.ipynb`.

H. Details in Neural Network

This appendix details the technical implementation of a neural network (NN) to predict optimized parameters for the BB84 Quantum Key Distribution (QKD) protocol with decoy states, based on input features derived from fiber length and detection events. The goal is to approximate the parameter optimization process described in Appendix **??**.

I. Data Preparation

The training data is sourced from `cleaned_combined_datasets.json`, containing entries grouped by n_X (detection events, 10^4 to 10^9):

- **Input Features (X)**: $[e_1, e_2, e_3, e_4]$, where $e_1 = L/100$, $e_2 = -\log_{10}(6\times10^{-7})$, $e_3 = 5\times10^{-3}\times100$, $e_4 = \log_{10}(n_X)$.
- **Target Outputs (Y)**: Optimized parameters $[\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X]$ from prior optimization.
- **Filtering**: Entries with key_rate ≤ 0 or $L > 200$ km are excluded.
- **Scaling**: X is standardized using `StandardScaler`, and Y is scaled to [0, 1] with `MinMaxScaler`. Scalers are saved as `scaler.pkl` and `y_scaler.pkl`.
- **Dataset Split**: 80% training (shuffled), 20% validation, using `torch.utils.data.random_split`.
- **DataLoader**: Batch size 64, shuffled for training, unshuffled for validation.

Evaluation data includes the full dataset and a separate test set for $n_X = 5\times10^8$ from `5e8_100_reordered_qkd_grouped_da`

J. Neural Network Architecture

The model (`BB84NN`) is a fully connected feedforward neural network implemented in PyTorch:

$$\text{Architecture: } 4 \xrightarrow{\text{ReLU}} 16 \xrightarrow{\text{ReLU}} 32 \xrightarrow{\text{ReLU}} 16 \xrightarrow{\text{Linear}} 5, \tag{56}$$

where:

- Input layer: 4 nodes (for e_1, e_2, e_3, e_4).
- Hidden layers: 16, 32, 16 nodes with ReLU activation.
- Output layer: 5 nodes (for $\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X$).

The model is deployed on the Metal Performance Shaders (MPS) backend (`torch.device("mps")`) for GPU acceleration on macOS.

K. Training Process

Training is conducted over 5000 epochs:

- **Loss Function**: Mean Squared Error (`MSELoss`).
- **Optimizer**: Adam (`optim.Adam`), initial learning rate 10^{-3}.
- **Scheduler**: `ReduceLROnPlateau`, reducing learning rate by 0.1 if validation loss plateaus for 5 epochs, minimum 10^{-6}.
- **Training Loop**: For each epoch, compute training and validation losses, update weights, and adjust learning rate.
- **Metrics**: Losses and learning rates are logged for analysis.

The model is saved as `bb84_nn_model.pth` after the first and last epochs.

L. Evaluation and Visualization

Evaluation occurs at epochs 1 and 5000, and post-training for $n_X = 5 \times 10^8$:

- **Prediction**: Model outputs scaled parameters, inverse-transformed via `y_scaler`, clipped to physical bounds ($\mu_1, \mu_2 \geq 10^{-6}$, probabilities in [0, 1]).
- **Key Rate Calculation**: Predicted parameters are fed into `safe_objective`, a wrapper for `objective` (Appendix **??**), returning 0 on NaN or errors.
- **Subplot Visualizations**:
  - **Key Rates**: Log-scaled optimized vs. predicted key rates vs. fiber length for $n_X = 10^4$ to 10^9, in 2x3 subplots, saved as `keyrate_subplots_*.png`.
  - **Parameters**: Optimized vs. predicted parameters vs. fiber length, in 2x3 subplots with a shared legend, saved as `parameters_subplots_*.png`.
- **Final Test**: For $n_X = 5 \times 10^8$, a combined plot of key rates and parameters is generated (`keyrate_parameters_5e8_final_test.png`).
- **Metrics**: Mean Absolute Error (MAE) is computed for parameters and key rates, excluding NaN values.

Loss curves (training and validation) and learning rates are plotted and saved as `loss_plot.png` and `learning_rate_plot.png`.

M. Implementation Details

- **Libraries**: PyTorch for NN, NumPy for data handling, Scikit-learn for scaling and metrics, Matplotlib (Agg backend) for plotting, Joblib for scaler serialization.
- **Error Handling**: NaN key rates are filtered, with diagnostics printed.
- **Codebase**: Available at `https://gits-15.sys.kth.se/slleung/QKD_KeyRate_ParameterOptimization/blob/main/Analysis/BB84_Parameters_2014_Analysis_Jax.ipynb`.

References

[1] Apple Inc. *Apple Silicon Overview*. Accessed: March 18, 2025. 2022. URL: https://www.apple.com/silicon-overview/.

[2] Apple Inc. *Metal Programming Guide*. Accessed: March 18, 2025. 2022. URL: https://developer.apple.com/metal/.

[3] Charles H Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: *Theoretical computer science* 560 (2014), pp. 7–11.

[4] Chi-Hang Fred Fung, Xiongfeng Ma, and HF Chau. "Practical issues in quantum-key-distribution postprocessing". In: *Physical Review A—Atomic, Molecular, and Optical Physics* 81.1 (2010), p. 012318.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Accessed: March 18, 2025. Cambridge, MA: MIT Press, 2016. ISBN: 9780262035613. URL: http://www.deeplearningbook.org/.

[6] Lester Ingber. "Simulated Annealing: Practice versus Theory". In: *Mathematical and Computer Modelling* 18.11 (1993). Accessed: March 18, 2025, pp. 29–57.

[7] Intel Corporation. *oneAPI Programming Guide*. Accessed: March 18, 2025. 2023. URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

[8] Charles Ci Wen Lim et al. "Concise security bounds for practical decoy-state quantum key distribution". In: *Physical Review A* 89.2 (2014), p. 022307.

[9] Weiqi Liu et al. "Integrating machine learning to achieve an automatic parameter prediction for practical continuous-variable quantum key distribution". In: *Physical Review A* 97.2 (2018), p. 022316.

[10] Weizhao Lu et al. "Recurrent neural network approach to quantum signal: coherent state restoration for continuous-variable quantum key distribution". In: *Quantum Information Processing* 17 (2018), pp. 1–14.

[11] John A Nelder and Roger Mead. "A simplex method for function minimization". In: *The computer journal* 7.4 (1965), pp. 308–313.

[12] NVIDIA Corporation. *CUDA Toolkit Documentation*. Accessed: March 18, 2025. 2023. URL: https://docs.nvidia.com/cuda/.

[13] Christoph Roser. *Local Optima Global Optimum*. Illustration. Available at: https://www.allaboutlean.com/ (accessed: March 04, 2025), Licensed under Creative Commons Attribution-ShareAlike 4.0 International License (CC-BY-SA 4.0), https://creativecommons.org/licenses/by-sa/4.0/. July 2018. URL: https://www.allaboutlean.com/.

[14] V. Scarani et al. "The Security of Practical Quantum Key Distribution". In: *Reviews of Modern Physics* 81 (2009), pp. 1301–1350.

[15] SciPy Developers. *SciPy Documentation: dual_annealing*. Accessed: March 18, 2025. 2023. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html.

[16] C. Tan et al. *Optimization Algorithms for Real-Time Systems*. Accessed: March 18, 2025. Amsterdam, Netherlands: Elsevier, 2018. ISBN: 9780123456789.

[17] TensorFlow Team. *TensorFlow Lite on ARM*. Accessed: March 18, 2025. 2023. URL: https://www.tensorflow.org/lite/guide.

[18] Wenyuan Wang and Hoi-Kwong Lo. "Machine learning for optimal parameter prediction in quantum key distribution". In: *Physical Review A* 100.6 (2019), p. 062334.

[19] Y. Xiang et al. "Simulated Annealing with GPU Acceleration". In: *IEEE Transactions on Parallel and Distributed Systems* 24.8 (2013). Accessed: March 18, 2025, pp. 1568–1577.