Universiteit
Leiden
The Netherlands

Bachelor Data Science & Artificial Intelligence

Annealing-Based QUBO Optimization
for NLP Hyperparameter Tuning

Roza Pilarek

Supervisor:
Dr. Hao Wang

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

Abstract

We study whether an *annealing-based, QUBO-driven* optimiser offers practical advantages over classical methods for hyperparameter optimization (HPO) in Natural Language Processing (NLP). We model a small HPO task, tuning learning rate, epochs, and batch size for DistilBERT on IMDb dataset into a compact QUBO with 16 binary variables. A Random Forest surrogate, trained on all 18 true configurations, provides a fixed, noise-free landscape on which we compare an annealing-based solver (D-Wave Ocean *SimulatedAnnealingSampler*) against Optuna/TPE (Bayesian optimization) and DEAP/GA (evolutionary algorithm) over 100 runs with equal resources.

The annealing-based solver found the global optimum in **100%** of runs with **zero variance**, while DEAP and Optuna achieved **82%** and **71%** success, respectively. Mean performances were near-identical (≈ 0.93) across methods, so *reproducibility* was the differentiator.

Statistical tests (Kruskal-Wallis, Mann-Whitney with Holm-Bonferroni) confirmed significant differences ($p < 0.001$), with small-to-medium effect sizes. However, annealing has higher costs: a one-time full evaluation of 18 configurations (∼5,400 seconds) and slower per-run search time (∼10.2 seconds) compared to DEAP (∼0.060 seconds) and Optuna (∼0.025 seconds). For this small space, annealing's setup cost is not offset, but it excels in small, discrete spaces reused often and requiring high reproducibility.

The contributions include (i) a systematic QUBO formulation for NLP HPO, (ii) a matched-budget benchmark against strong classical baselines on a fixed surrogate landscape, (iii) evidence-based guidance on when to prefer annealing versus classical search, and (iv) a fair analysis of scalability limitations. All experiments use a *classical* annealer. Implications for quantum hardware (noise, embedding, precision) are discussed.

Contents

1 Introduction

1.1 Background and Motivation

In Natural Language Processing (NLP), the performance of deep learning models heavily depends on choosing the right hyperparameters. These configuration choices, including learning rate, batch size, and number of epochs, influence accuracy, training time, and generalization capabilities of the model. Still, hyperparameter optimization (HPO) is often one of the most time-consuming and resource-intensive parts of model development.
Traditional HPO methods, such as grid search and random search [BB12], become impractical as models grow larger, especially for transformer-based models like BERT [DCLT19]. Each evaluation can take a long time, and the number of possible configurations increases rapidly. Advanced classical methods, like Bayesian optimization or evolutionary algorithms, are more efficient but can produce inconsistent results across runs.
Quantum annealing [KN98], offered by systems like D-Wave [Sys], provides a different approach. It is designed to find optimal solutions for combinatorial problems by formulating them as Quadratic Unconstrained Binary Optimization (QUBO) problems [Glo22]. Quantum annealers use quantum effects, like tunneling, to explore solutions, potentially avoiding getting stuck in suboptimal results. However, there is little evidence on how well annealing works for NLP HPO, including how to encode the problem, how it compares to classical methods, and its computational costs.
In this work, we evaluate an annealing-based QUBO solver using D-Wave's *SimulatedAnnealingSampler* (a classical annealer) as a proxy, implications of which are discussed later.

1.2 Problem Statement

Modern transformer models require tuning mixed-discrete hyperparameters whose effects interact non-linearly. Each evaluation is costly, so HPO is both compute-intensive and operationally constraining. We encode NLP HPO as a QUBO and compare an *annealing-based* solver to strong classical baselines (Optuna/TPE, DEAP/GA) under matched budgets, assessing solution quality, efficiency, and reproducibility.

1.3 Research Questions

This thesis investigates whether annealing-based QUBO optimization can offer practical advantages for NLP hyperparameter tuning. In our experiments we use D-Wave's classical *SimulatedAnnealingSampler* as a proxy for annealing behaviour.

Main research question.

> **Can an annealing-based QUBO solver outperform classical methods for NLP hyperparameter tuning in terms of accuracy, efficiency, and consistency?**

To address this, we formulate four focused sub-questions.

Sub-question 1: QUBO formulation

> **How can NLP hyperparameter optimization be formulated as a Quadratic Unconstrained Binary Optimization (QUBO)?**

We create a clean mapping that uses binary encodings for discrete choices, auxiliary variables for interactions, and penalty terms that enforce the validity of such combinations [Glo22].

Sub-question 2: Model performance

> **How do the best configurations found by annealing compare to those found by classical methods on an NLP task?**

We evaluate DistilBERT fine-tuning on IMDb, comparing the *true* validation accuracy of the selected configurations and the success rate in finding the global optimum.

Sub-question 3: Computational requirements

> **How do the time and resources needed for annealing compare to classical methods, and how do they scale?**

We measure one-time setup cost, marginal per-run cost, number of true evaluations and surrogate queries, and wall-clock time, analyzing how they scale [KYN+15].

Sub-question 4: Optimization consistency

> **Is annealing more consistent across runs than classical methods, and by how much?**

We assess consistency by measuring the variability in best configurations and success rates across repeated runs.

1.4 Research Objectives

To answer these questions, we set four clear objectives.

Objective 1: Develop QUBO Formulation The first objective is to design and implement a QUBO model for NLP HPO. This includes (i) binary encodings for discrete/categorical choices, (ii) auxiliary variables to capture interactions between terms, (iii) penalty terms to enforce valid results, and (iv) a brief scalability analysis [Glo22].

Objective 2: Implement a Comparable Evaluation Framework The second objective includes creating an evaluation framework to compare an annealing-based solver with classical baselines. This includes an annealing pipeline using D-Wave Ocean SDK (*SimulatedAnnealingSampler* in our experiments) [Sys], Bayesian Optimization [SLA12, ASY+19] and an Evolutionary Algorithm [FDRG+12], choosing evaluation metrics, and designing statistical tests for comparison.

Objective 3: Empirical Evaluation on Real NLP Task The third objective focuses on conducting experiments on a representative NLP task: sentiment analysis on the IMDb dataset [MDP+11] with DistilBERT [SDCW20]. It also includes tuning learning rate, epochs, and batch size, and reporting both surrogate-level outcomes and *true* validation performance for selected configurations.

Objective 4: Provide Practical Guidelines The final objective is to develop practical recommendations: when annealing-based QUBO search is attractive (e.g., small-to-medium discrete spaces, strong reproducibility requirements), how to formulate a QUBO, and when classical methods are preferable.

1.5 Scope and Limitations

1.5.1 In Scope

This study evaluates hyperparameter optimization (HPO) for a single, representative NLP task:

- **Task & data.** Binary sentiment classification on IMDb [MDP+11]. We use the official 25,000/25,000 train/test split and derive a fixed 90/10 split from the 25,000 train portion for training/validation (seed=42). The test set remains untouched until final evaluation.
- **Model.** DistilBERT (base, uncased) [SDCW20] fine-tuning with standard training settings, measuring *validation accuracy* as the main metric.
- **Search space.** Three discrete hyperparameters: learning rate $\in \{1e{-}5, 3e{-}5, 5e{-}5\}$, epochs $\in \{2, 3, 4\}$, batch size $\in \{16, 32\}$, resulting in $3 \times 3 \times 2 = 18$ configurations.
- **Optimisers.**
  1. *Annealing-based QUBO solver* implemented with D-Wave Ocean's *SimulatedAnnealingSampler* [Sys].
  2. *Bayesian optimization* via Optuna (TPE) [SLA12, ASY+19].
  3. *Evolutionary algorithm* via DEAP (GA) [FDRG+12].
- **Surrogate usage.** A Random Forest surrogate [Bre01] is trained on *all 18* ground-truth evaluations to define a fixed landscape shared by all optimisers. This fixed landscape isolates optimiser behaviour from model-training noise.

1.5.2 Limitations

Simulated annealing, not QPU. The experiments use a classical simulated annealer (not a quantum processing unit). This means that results do not capture QPU-specific effects such as embedding/chain breaks, analog noise, and coefficient precision limits [RWJ+14].

Small discrete space. The 18-point space allows full evaluation but is smaller than many real-world HPO problems.

Discrete focus. We tune only three discrete hyperparameters, omitting continuous or larger spaces, which may affect optimiser behaviour.

Surrogate approximation. Although trained on all 18 points, a surrogate can still introduce small approximation error.

1.6 Thesis Structure

The remainder of this thesis is organized as follows.

Section 2: Background and Theory. Covers HPO methods, quantum and annealing-based optimization, QUBO formulation, NLP basics, and related work.

Section 3: Methodology. Describes the experimental setup, including the NLP task, search space, QUBO design, surrogate model, optimiser pipelines, and statistical analysis.

Section 4: Results. Presents findings, including baseline performance, optimiser comparisons, consistency, computational costs, and statistical tests.

Section 5: Discussion. Analyzes results, explains when annealing excels or falls short, compares to prior work, and discusses limitations.

Section 6: Conclusions. Summarizes contributions, evaluates annealing-based QUBO for NLP HPO, and suggests future research directions.

2 Background and Theory

2.1 Hyperparameter Optimization in Machine Learning

Hyperparameters are settings that control how a machine learning model learns, but they are not learned from the data itself. Examples in deep learning and NLP include learning rate, batch size, number of epochs, dropout rate, and model architecture choices like the number of layers. These settings greatly affect model accuracy, training time, and how well the model generalizes to new data.
The goal of hyperparameter optimization (HPO) is to find the best configuration, defined as:

$$\theta^* = \arg\min_{\theta \in \Theta} \ \mathcal{L}(M_\theta, D_{\text{val}}) \tag{1}$$

where θ is a hyperparameter configuration, Θ is the search space, M_θ is the model trained with θ, and $\mathcal{L}$ is the validation loss on D_{val}. If accuracy is the target, the objective becomes $\arg\max_{\theta \in \Theta} \ \text{Acc}(M_\theta, D_{\text{val}})$.

2.1.1 Classical Optimization Methods

Grid Search. Grid search tests every possible combination of predefined hyperparameter values. For k hyperparameters with n_i values each, it requires $\prod_{i=1}^{k} n_i$ evaluations. It is straightforward and guarantees finding the best configuration in the grid, but it becomes impractical as the number of combinations grows, struggles with continuous variables, and does not learn from previous trials.

Random Search. Random search samples configurations randomly within a fixed budget [BB12]. It often performs better than grid search when only a few hyperparameters matter most, as it covers the space more broadly. However, it offers no guarantee of finding the best solution and does not use past results to guide future samples.

Bayesian Optimization. Bayesian optimization (BO) models HPO as a sequential decision process using a surrogate model to predict performance [SLA12]. Common surrogates include Gaussian Processes or Tree-structured Parzen Estimators (TPE, used by Optuna [ASY+19]). An acquisition function, like Expected Improvement (EI) or Upper Confidence Bound (UCB) balances exploring new areas and exploiting known good ones. BO is efficient and provides uncertainty estimates, but maintaining the surrogate adds overhead, and it can struggle with many discrete or categorical variables or high-dimensional spaces.

Evolutionary Algorithms. Evolutionary algorithms (EAs) evolve a population of candidate solutions through selection, crossover, and mutation [FDRG+12]. They handle discrete and mixed-type variables well, maintain diverse solutions, and can escape suboptimal solutions via mutation. However, they need tuning of their own parameters (e.g., population size), often require more evaluations, and lack strong theoretical guarantees.

2.2 Quantum Annealing Fundamentals

2.2.1 Quantum Computing Paradigms

Quantum computing uses quantum phenomena like superposition, entanglement, and interference to process information differently from classical computers. *Gate-based quantum computing* uses quantum logic gates to manipulate qubits, supporting a wide range of algorithms (e.g., Shor's or Grover's). *Adiabatic quantum computing (AQC)* relies on slowly evolving a quantum system to stay in its lowest-energy state [FGG+01]. *Quantum annealing (QA)* is a specialized form of AQC focused on optimization [KN98].

2.2.2 The Quantum Annealing Process

Quantum annealing explores an energy landscape using a time-dependent Hamiltonian:

$$H(t) = A(t)\, H_0 + B(t)\, H_P, \quad t \in [0, T], \tag{2}$$

where H_0 is a "driver" Hamiltonian with an easy ground state, and H_P is the problem Hamiltonian that encodes the objective (typically an Ising model).

An Ising model assigns each variable a spin $s_i \in \{-1, +1\}$ and defines an energy $H_{\text{Ising}} = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j$, where h_i are local fields and J_{ij} are couplings. Minimizing this energy selects spin alignments (negative J_{ij} favors alignment, positive favors anti-alignment).
The schedule functions satisfy $A(0) \approx 1$, $B(0) \approx 0$ and $A(T) \approx 0$, $B(T) \approx 1$ [KN98]. If the evolution is slow enough, the system stays in the ground state, finding the optimal solution. Unlike classical methods that climb over barriers, QA can "tunnel" through narrow barriers, potentially avoiding local minima [KN98].
Note. In this thesis we evaluate an *annealing-based* solver using D-Wave's *SimulatedAnnealingSampler* (classical) on QUBO instances. Implications for quantum hardware (noise, embedding, precision) are discussed later.

2.2.3 QUBO Formulation

Quadratic Unconstrained Binary Optimization (QUBO) is the standard format for annealers [Luc14]. The goal is to minimize:

$$\min_{x \in \{0,1\}^n} \; f(x) \;=\; x^{\mathsf{T}} Q x \;=\; \sum_i Q_{ii}\, x_i \;+\; \sum_{i<j} Q_{ij}\, x_i x_j, \tag{3}$$

with binary vector x and coefficient matrix Q. QUBO is equivalent to the Ising form via $s_i = 2x_i - 1$,

$$H_{\text{Ising}} \;=\; \sum_i h_i\, s_i \;+\; \sum_{i<j} J_{ij}\, s_i s_j. \tag{4}$$

Constraints are enforced by adding quadratic penalties. For an equality constraint $g(x) = 0$,

$$f_{\text{constrained}}(x) \;=\; f(x) \;+\; P\, g(x)^2, \tag{5}$$

where P is large enough to prevent invalid solutions [Glo22]. Common encodings include one-hot selections and auxiliary variables to keep interactions quadratic.

2.3 Natural Language Processing and Deep Learning

2.3.1 Transformer Architecture

The Transformer [VSP+17] uses self-attention and feed-forward networks instead of recurrence or convolution. Its key components are:

- **Multi-head self-attention** (mixes context across positions),
- **Positional encodings/embeddings** (adds sequence order information),
- **Position-wise feed-forward networks** (applies nonlinear transformation),
- **Residual connections** and **LayerNorm** (improves training stability).

Scaled dot-product attention is denoted as

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\!\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \tag{6}$$

2.3.2 BERT and DistilBERT

BERT [DCLT19] pretrains a bidirectional Transformer with masked language modeling and next-sentence prediction. Common sizes include `base` (12 layers, hidden 768, 12 heads, ∼110M params) and `large` (24 layers, hidden 1024, 16 heads, ∼340M).
DistilBERT [SDCW20] distills BERT to a smaller model (6 layers, hidden 768, 12 heads, ∼66M), removing token-type embeddings and NSP while retaining ∼97% of BERT's performance with ∼40% fewer parameters and ∼60% higher speed.

2.3.3 Sentiment Analysis Task

Sentiment analysis classifies text as positive or negative. Metrics include accuracy, F1, precision, and recall. Applications span product reviews, social media, and customer feedback, and challenges include sarcasm, domain shift, and context dependence [MDP+11]. In this work we use binary sentiment classification on the IMDb dataset.

2.3.4 Hyperparameters in NLP Models

Fine-tuning transformers includes:

- **Training:** learning rate (step size), batch size (memory/gradient noise trade-off), and epochs (under/over-fitting balance).
- **Regularization:** dropout rate, weight decay, gradient clipping.
- **Architecture:** depth (layers), hidden size, number of heads.
- **Optimization:** optimiser choice.

These choices interact (e.g., epoch count depends on learning rate and batch size), which is important to note for later QUBO formulation.

2.4 Surrogate Modeling

2.4.1 Concept and Applications

Surrogate modeling involves learning a cheap approximation to an expensive objective. In HPO, where each evaluation may require training a deep network, a surrogate can reduce cost by predicting validation performance for candidate hyperparameters before full training begins.
A generic surrogate workflow comprises four stages: (i) *Initial sampling*: evaluate the true objective at selected configurations, (ii) *Model fitting*: train the surrogate on observed pairs (θ, y), (iii) *Acquisition*: use the surrogate to propose promising configurations (balancing exploration/exploitation), (iv) *Update*: incorporate new observations and refit (for iterative schemes).
Common surrogates include Gaussian processes (good uncertainty but poor scaling), Random Forest (robust on small, mixed-type data) [Bre01], and neural networks (flexible but data-hungry).

2.4.2 Random Forest as Surrogate

Random Forest (RF) averages predictions from B decision trees:

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x), \tag{7}$$

where T_b denotes the b-th tree trained on a bootstrap sample. RFs naturally handle discrete and mixed variables, capture nonlinear interactions, are robust to outliers, and are fast to evaluate. Although RFs lack a principled Bayesian posterior, the variance across trees can serve as a rough uncertainty estimate:

$$\sigma^2(x) = \frac{1}{B} \sum_{b=1}^{B} \big(T_b(x) - \hat{f}(x)\big)^2, \tag{8}$$

noting that this quantity is typically uncalibrated.

This thesis' setup. We adopt a *static* (offline) surrogate: evaluate *all 18* configurations once to obtain ground truth, train a single RF on these data, and use that surrogate as a fixed landscape for *all* optimisers (annealing-based QUBO solver, Optuna/TPE, and DEAP/GA). This isolates differences due to search strategies rather than noise from repeated model training. Final reported performance for chosen configurations is also mapped back to the *true* validation accuracies to check for surrogate-induced ranking errors.

2.4.3 Trade-offs in Surrogate-based Optimization

Surrogates introduce several trade-offs:

Accuracy vs speed. Predictions are approximate (risking near-tie inversions), but they are orders of magnitude cheaper than true evaluations. Accuracy improves with more observations.

Exploration vs exploitation. Uncertainty guides exploration, but over-reliance can lead to early convergence.

Online vs offline. Online (iterative) surrogates adapt as new data arrive (typical in BO). Offline (static) surrogates, as used here, provide a fixed landscape for controlled comparisons.

Objective shaping. To integrate with QUBO minimization, we minimize $-$Accuracy) and scale coefficients so constraint penalties dominate any illegal gain.

Scope limitations. A surrogate trained on a small discrete domain may not generalize beyond that domain. Here, this is intentional to create a controlled sandbox.
A surrogate allows tractable and *fair* comparisons between optimization strategies by holding the objective landscape fixed. We leverage this to compare annealing-based QUBO optimization with the classical baselines under matched budgets, then verify selected configurations against true validation scores.

2.5 Related Work

2.5.1 Annealing for Machine Learning

Early applications of annealing for ML framed learning tasks as QUBO/Ising problems and used D-Wave hardware to search for low-energy solutions. Mott et al. [Mot17] formulated a high-energy-physics classification pipeline with a QUBO and demonstrated that annealing could reach high-quality solutions on small instances. However, they highlighted practical challenges: the impact of limited coefficient precision ("limited precision problem"), small problem sizes driven by hardware embeddings, and extra work needed to match the hardware's structure.
Later studies and tutorials on QUBO modeling [Luc14, Glo22] explained how to model choices and simple relationships using quadratic penalties. Other research [RWJ+14, KYN+15] showed that claims of quantum benefits need careful evaluation, considering issues like noise, problem mapping, scaling of numbers, and fair comparisons with classical methods.

2.5.2 Research Gap Analysis

Most studies using annealing focus on simple or artificial ML tasks, not modern NLP. Three gaps motivate this study:

1. **Task realism for NLP HPO.** There is little evidence of annealing being used for practical NLP tasks with clear search spaces and evaluation metrics.

2. **Fair comparisons.** Few studies compare annealing to strong classical methods (e.g., Bayesian optimization [SLA12, ASY+19] or evolutionary algorithms [FDRG+12]) using the same resources and proper statistical tests.

3. **Operational guidance.** There is no clear advice on when annealing is better than classical methods, especially regarding consistency, setup effort, number scaling, and hardware-specific issues like chains or embeddings.

This thesis. We address these gaps by: (i) creating a clear QUBO model for an NLP task (optimizing learning rate, epochs, and batch size), (ii) comparing an *annealing-based QUBO solver* (D-Wave Ocean's *SimulatedAnnealingSampler*) to classical methods (Optuna/TPE and DEAP/GA) with equal resources using statistical tests, and (iii) reporting results on success rates, variability, and costs, with non-parametric tests and effect sizes. Although our experiments use a classical annealer (not a QPU), the QUBO model and findings apply to quantum hardware, and we discuss differences (e.g., noise, chain strength, and precision) based on prior studies [RWJ+14, KYN+15].

3 Methodology

3.1 Experimental Design Overview

Our goal is to compare an annealing-based QUBO solver against classical HPO methods on a realistic NLP task. The workflow has five phases:

(1) Problem definition. We set the task (IMDb binary sentiment [MDP+11]) and model (DistilBERT [SDCW20]). The hyperparameter space is discrete: learning rate $\in \{1e{-}5, 3e{-}5, 5e{-}5\}$, epochs $\in \{2, 3, 4\}$, batch size $\in \{16, 32\}$ (18 configurations total). The primary selection metric is *validation accuracy*.

(2) Ground truth and data protocol. Using the official IMDb split, we derive a fixed 90/10 train/validation split from the 25,000 training set (seed=42). The 25,000 test set remains untouched until final checks. We train/evaluate *all 18* configurations once to obtain true validation scores.

(3) Static surrogate. We train a Random Forest regressor [Bre01] on the 18 pairs to obtain a *fixed* surrogate landscape shared by all optimisers. This isolates differences due to search strategies rather than noisy retraining.

(4) Optimization pipelines under matched budgets.

- **Annealing-based QUBO solver** (D-Wave Ocean *SimulatedAnnealingSampler*): construct a QUBO from surrogate scores (minimize −Accuracy plus penalties) and sample with `num_reads`= 500.
- **Bayesian Optimization** (Optuna/TPE): `n_trials`= 25.
- **Evolutionary Algorithm** (DEAP/GA): population= 10, generations= 5 ($\approx$ 60 surrogate evaluations).

Each method is repeated 100 times with controlled seeds. Setup cost (18 true evaluations, surrogate fit, and QUBO build) is reported *separately* from per-run optimization time.

(5) Evaluation and statistics. We report success rate in finding the global optimum, mean and variance of best-found surrogate scores, wall-clock time, and the number of evaluations.

3.2 NLP Task and Dataset

3.2.1 IMDb Movie Review Dataset

We use the IMDb movie review dataset [MDP+11] as a balanced, binary sentiment benchmark with 50,000 labeled reviews (25,000 positive, 25,000 negative). We keep the official 25,000 test split untouched for final evaluation. From the 25,000 train split, we derive a fixed **90/10 train/validation** split (seed = 42), resulting in **22,500** training and **2,500** validation examples.
Tokenization uses the DistilBERT tokenizer from `transformers` [WDS+20] with truncation to 512 subword tokens and static padding:

```
from datasets import load_dataset, DatasetDict
from transformers import DistilBertTokenizerFast

# 1) Load and split (90/10 from the 25,000 train portion)
raw = load_dataset("imdb")
split = raw["train"].train_test_split(test_size=0.1, seed=42)
```

```
ds = DatasetDict({
    "train_subset": split["train"],
    "val_subset":   split["test"],
    "test":         raw["test"],
})

# 2) Tokenizer
tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-
    uncased")

def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        padding="max_length",
        max_length=512,
    )

# 3) Tokenize all splits (keep 'label', drop raw 'text' to save RAM)
ds_enc = ds.map(tokenize_function, batched=True, remove_columns=["text"
    ])
```

Listing 1: Tokenization with DistilBertTokenizerFast

3.2.2 DistilBERT Model

We fine-tune DistilBERT [SDCW20]. DistilBERT uses GeLU activations and learned positional embeddings. We report *validation accuracy* as the primary selection metric.
Fine-tuning setup (key hyperparameters are supplied from the HPO loop):

```
from transformers import (
    DistilBertForSequenceClassification,
    TrainingArguments, Trainer
)

model = DistilBertForSequenceClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=2
)

training_args = TrainingArguments(
    output_dir="...",
    learning_rate=learning_rate,              # {1e-5,3e-5,5e-5}
    num_train_epochs=epochs,                  # {2,3,4}
    per_device_train_batch_size=batch_size,  # {16,32}
    per_device_eval_batch_size=batch_size*2,
    weight_decay=0.01,
    evaluation_strategy="epoch",
```

```
    logging_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    save_total_limit=1,
    seed=42,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=ds_enc["train_subset"],    # 22,500
    eval_dataset=ds_enc["val_subset"],       # 2,500
    tokenizer=tokenizer
)
```

Listing 2: DistilBERT fine-tuning with Trainer

The held-out **25,000 test set** is used only once, to report the final performance of the best configuration(s) selected by each optimiser.

3.3 Hyperparameter Space Definition

3.3.1 Selected Hyperparameters

We tune three hyperparameters with strong effects on BERT-family fine-tuning:

- **Learning rate (LR)** $\in \{1\times10^{-5}, 3\times10^{-5}, 5\times10^{-5}\}$. These values follow a log-scale range to balance stability and convergence speed [DCLT19].
- **Epochs** $\in \{2, 3, 4\}$. 2-4 epochs are commonly sufficient for GeLU/IMDb-style tasks. More epochs can overfit for these models/datasets.
- **Batch size** $\in \{16, 32\}$. These sizes are commonly used for fine-tuning, balancing memory footprint and gradient noise.

This results in $3 \times 3 \times 2 = \mathbf{18}$ discrete configurations.

3.4 QUBO Formulation and Design

3.4.1 Binary variables and encoding

We encode three discrete hyperparameters with binary variables:

- Learning rate: one-hot $(x_{\text{lr1}}, x_{\text{lr3}}, x_{\text{lr5}})$ for $\{1\text{e}{-5}, 3\text{e}{-5}, 5\text{e}{-5}\}$ with $\sum x_{\text{lr}\cdot} = 1$.
- Epochs: one-hot $(x_{\text{e2}}, x_{\text{e3}}, x_{\text{e4}})$ with $\sum x_{\text{e}\cdot} = 1$.
- Batch size: a single bit x_{bs} with $0 \leftrightarrow 16$ and $1 \leftrightarrow 32$.

To capture LR×Epochs interactions, we introduce nine auxiliary variables y_{ij}, one for each pair ($\text{lr} \in \{1,3,5\}, \text{ep} \in \{2,3,4\}$), intended to equal $x_{\text{lr}i}\, x_{\text{e}j}$. This results in total logical variables: $3+3+1+9=16$.

3.4.2 Objective from the surrogate

Let $\widehat{\text{Acc}}(\text{lr}, \text{ep}, \text{bs})$ be the Random Forest surrogate prediction. We minimize the negated score so higher accuracy means lower energy. Using $x_{\text{bs}} \in \{0,1\}$:

$$f_{\text{obj}} = \sum_{i,j} \Big[\underbrace{-\widehat{\text{Acc}}(\text{lr}_i, \text{ep}_j, 16)}_{\text{coefficient of } y_{ij}}\; y_{ij} \;-\; \underbrace{\big(\widehat{\text{Acc}}(\text{lr}_i, \text{ep}_j, 32) - \widehat{\text{Acc}}(\text{lr}_i, \text{ep}_j, 16)\big)}_{\text{coefficient of } y_{ij}x_{\text{bs}}}\; y_{ij}\, x_{\text{bs}} \Big].$$

This contributes linear terms on y_{ij} and quadratic couplings $y_{ij}\, x_{\text{bs}}$.

3.4.3 Constraints via quadratic penalties

Exactly one LR and one Epoch. For each one-hot group,

$$P_{\text{lr}}\Big(\sum_k x_{\text{lr}k} - 1\Big)^2 \;+\; P_{\text{ep}}\Big(\sum_k x_{\text{e}k} - 1\Big)^2.$$

Each expands (using $x^2 = x$) to linear $-P$ on each bit and pairwise $+2P$ on distinct pairs.

Auxiliary product consistency. For each pair (i,j) enforcing $y_{ij} = x_{\text{lr}i} x_{\text{e}j}$,

$$P_{ij}\Big(x_{\text{lr}i}\, x_{\text{e}j} - 2x_{\text{lr}i}\, y_{ij} - 2x_{\text{e}j}\, y_{ij} + 3y_{ij}\Big).$$

3.4.4 Complete QUBO and coefficient selection

The full QUBO is the sum of f_{obj} and all penalties, written as $x^\top Q x$ with $x \in \{0,1\}^{16}$ and symmetric Q. We keep coefficients in a compact range and choose penalties so constraint violations are not energetically preferred. In practice we used $P_{\text{lr}} = P_{\text{ep}} = P_{ij} = 2.0$, which exceeded the largest objective delta from any single flip on this landscape. This method yielded no decoding errors under simulation.

3.4.5 Decoding and validity

A solution is decoded by selecting the active LR and Epoch bits (each exactly one), mapping x_{bs} to $\{16, 32\}$, and optionally checking y_{ij} consistency. Invalid decodes indicate penalties that are too small (or, on hardware, insufficient chain strength). These were not observed in our simulated runs.

3.5 Surrogate Model Development

3.5.1 Random Forest Regressor

We use a Random Forest (RF) regressor [Bre01] to approximate the expensive fine-tuning objective. Because our search space has only 18 points, we first obtain *ground-truth* validation accuracies by training DistilBERT for every configuration (fixed 90/10 train/validation split, seed= 42). The RF then learns a mapping from hyperparameter tuples to validation accuracy.

Data protocol and encoding. The surrogate is trained only on validation accuracies. Categorical inputs are encoded with a fixed *ordinal* scheme using predefined category orders that match the discrete choices:

$$\text{LR} \in \{\texttt{1e-5}, \texttt{3e-5}, \texttt{5e-5}\}, \quad \text{Epochs} \in \{2, 3, 4\}, \quad \text{Batch} \in \{16, 32\}.$$

This results in a feature matrix of shape $(18, 3)$. Random seeds are fixed for reproducibility.

Model configuration.

```
from sklearn.ensemble import RandomForestRegressor
surrogate = RandomForestRegressor(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
# X: ordinal-encoded (LR, Epochs, Batch), shape (18, 3)
# y: true validation accuracies, shape (18,)
```

Listing 3: Random Forest surrogate configuration (scikit-learn)

Training data. The training set contains one row per configuration with its true validation accuracy. We keep the final 25,000 IMDb test set untouched until reporting final performance of selected configurations.

3.5.2 Performance Metrics

The surrogate achieves high fidelity on this domain:

- MSE $= \mathbf{8.58 \times 10^{-7}}$, MAE $= \mathbf{7.54 \times 10^{-4}}$, $R^2 = \mathbf{0.9495}$, Max Error $= \mathbf{0.0020}$.
- 5-fold cross-validation (on 18 points) shows low error variability:
  - Fold MSEs: [2.93×10^{-5}, 4.11×10^{-6}, 8.15×10^{-6}, 4.13×10^{-6}, 7.13×10^{-7}]
  - Fold MAEs: [0.004986, 0.001750, 0.002706, 0.001952, 0.000675]

The surrogate's absolute errors (0.002) are small relative to the accuracy range across configurations (≈ 0.014), which makes it suitable as a fixed landscape for comparing optimisers.

Feature importance. Permutation/impurity importance indicates learning rate contributes $\sim$45.2%, epochs $\sim$38.7%, and batch size $\sim$16.1% to the surrogate's predictions.

3.5.3 Operationalization for QUBO and Baselines

For the annealing-based solver, we create a set of predicted scores for all 18 configurations and adjust them using a simple transformation (minimizing $-$Accuracy) to fit the QUBO format. We keep the coefficients in a tight range and set penalty scales high enough to ensure constraint violations are heavily penalized. The same trained surrogate is used by Optuna (TPE) and DEAP (GA), so all methods work on the same, noise-free landscape. For the final selections, we report the *true* validation accuracy by mapping the chosen configuration back to the ground-truth table.

3.6 Optimization Pipelines

All methods optimize *the same, fixed* surrogate landscape (RF trained on all 18 true points). We report the one-time setup cost (true runs, surrogate fit, and QUBO build) separately from per-run tuning time.

3.6.1 Annealing-based QUBO Pipeline (Simulated)

We implement an annealing-based solver using D-Wave Ocean's *SimulatedAnnealingSampler* (classical simulated annealing over QUBO) [Sys]. The pipeline:

1. **QUBO construction:** From surrogate predictions for all 18 configurations, form $f_{\text{obj}}(\theta) = -\widehat{\text{Acc}}(\theta)$ plus quadratic penalties (one-hot and $y = ab$) with $P_{\text{lr}} = P_{\text{ep}} = P_{ij} = 2.0$ (chosen to exceed the largest objective delta from any single flip).

2. **Sampling:** Call `sample_qubo(Q, num_reads=500, seed=`s`)` to draw samples, s is varied per run for independence.

3. **Decoding & validity:** Select the lowest-energy *valid* sample (exactly-one LR/Epoch, y_{ij} consistent). Decode to $(\text{lr}, \text{epochs}, \text{batch})$.

4. **Scoring:** Report the surrogate score of the decoded config. Map it back to the true table when needed.

```python
from dwave.samplers import SimulatedAnnealingSampler

def anneal_optimize(qubo_dict, num_reads=500, seed=42):
    sampler = SimulatedAnnealingSampler()
    resp = sampler.sample_qubo(qubo_dict, num_reads=num_reads, seed=seed)
    best = resp.first.sample   # lowest-energy assignment
    cfg  = decode_dwave_solution(best)  # checks one-hot & y=ab consistency
    return cfg  # (lr, epochs, batch)
```

Listing 4: Annealing over QUBO (SimulatedAnnealingSampler)

On real QPUs, one would also tune chain strength and consider gauges. Here we use the classical sampler, which returned zero invalid decodes with $P = 2.0$.

3.6.2 Classical Optimization Pipelines

Optuna (Bayesian optimization, TPE). We use Optuna's TPE sampler [ASY+19] for a budget of **25 trials** per run. The objective minimizes the negative surrogate accuracy.

```
import optuna

def optuna_optimize(surrogate_predict, n_trials=25, run_seed=42):
    def objective(trial):
        lr     = trial.suggest_categorical("learning_rate", ["1e-5","3e-5","5e-5"])
        epochs = trial.suggest_categorical("epochs", [2,3,4])
        batch  = trial.suggest_categorical("batch_size", [16,32])
        acc = surrogate_predict(lr, epochs, batch)
        return -acc  # minimize negative accuracy
    study = optuna.create_study(
        sampler=optuna.samplers.TPESampler(seed=run_seed), direction="minimize"
    )
    study.optimize(objective, n_trials=n_trials, show_progress_bar=False)
    return study.best_params, -study.best_value
```

Listing 5: Optuna (TPE) on the surrogate

DEAP (Genetic algorithm). We use a small GA (population = 10, generations = 5, $\approx$60 surrogate evaluations) with two-point crossover and mutation probability of 0.2 per gene.

```
from deap import base, creator, tools, algorithms
import random

if not hasattr(creator, "FitnessMax"):
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
if not hasattr(creator, "Individual"):
    creator.create("Individual", list, fitness=creator.FitnessMax)

def deap_optimize(surrogate_predict, pop_size=10, n_gen=5, seed=42):
    random.seed(seed)
    toolbox = base.Toolbox()
    toolbox.register("attr_lr",    random.randint, 0, 2)
    toolbox.register("attr_epoch", random.randint, 0, 2)
    toolbox.register("attr_bs",    random.randint, 0, 1)
    toolbox.register("individual", tools.initCycle, creator.Individual,
                     (toolbox.attr_lr, toolbox.attr_epoch, toolbox.attr_bs), n=1)
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)
    def decode(ind):
        lrs = ["1e-5","3e-5","5e-5"]; eps=[2,3,4]; bs=[16,32]
```

```
        return (lrs[ind[0]], eps[ind[1]], bs[ind[2]])
    def evaluate(ind):
        lr,epochs,batch = decode(ind)
        return (surrogate_predict(lr,epochs,batch),)
    def mutate(ind, indpb=0.2):
        for i in range(len(ind)):
            if random.random() < indpb:
                ind[i] = [toolbox.attr_lr, toolbox.attr_epoch, toolbox.
                   attr_bs][i]()
        return (ind,)
    toolbox.register("evaluate", evaluate)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", mutate)
    toolbox.register("select", tools.selTournament, tournsize=3)

    pop = toolbox.population(n=pop_size)
    hof = tools.HallOfFame(1)
    algorithms.eaSimple(pop, toolbox, cxpb=0.7, mutpb=0.3, ngen=n_gen,
                        halloffame=hof, verbose=False)
    best = hof[0]
    return decode(best), best.fitness.values[0]
```

Listing 6: DEAP (GA) on the surrogate

Budgets, seeds, and fairness. Each method runs **100** independent repetitions with controlled seeds. Budgets are predeclared and appropriate to each method's mechanics: Optuna 25 trials, DEAP ≈60 surrogate evaluations (pop=10, gen=5), and annealing one QUBO solve per run built from all 18 surrogate scores with `num_reads`=500. All pipelines query the *same* surrogate to isolate optimiser behaviour from training noise.

3.7 Evaluation Metrics

Primary metrics.

- **Best validation accuracy (per run).** The accuracy of the configuration chosen by the optimiser. We report both (i) the *surrogate-predicted* best score and (ii) the corresponding *true* validation accuracy obtained from the ground-truth table.
- **Success rate.** The percentage of runs that find the best possible configuration out of the 18 true options. We report the estimate and a 95% Wilson confidence interval for the proportion.
- **Consistency.** How much the best scores vary across runs, measured by standard deviation (SD), coefficient of variation (CV = SD/mean), interquartile range (IQR), and median absolute deviation (MAD).

Secondary metrics

- **Tuning time.** Wall-clock time, split into *setup* (18 true evaluations, surrogate fit, and QUBO build) and *per-run search* (optimiser-only).
- **Configuration diversity.** Number of distinct configurations found across runs and their *entropy* (higher entropy means more varied exploration).
- **Convergence behaviour (for iterative methods).** Tracks the best score over trials or generations, summarized by (i) time-to-first-optimum (trials until the best configuration is found) and (ii) area under the improvement curve (normalized AUC).
- **Ground-state probability (for annealing).** The percentage of results at the lowest QUBO energy, indicating solution stability.

3.8 Statistical Analysis Methods

3.8.1 Assumption checks

Before parametric tests we assess:

- **Normality:** Use Shapiro-Wilk and Jarque-Bera tests on the best scores for each method.
- **Homoscedasticity:** Use Levene's test across methods.

Given simulated annealing's zero variance in our setup, these assumptions usually fail, which makes us prefer rank-based tests.

3.8.2 Comparisons across methods

- **Continuous outcomes (best scores).** Use the Kruskal-Wallis test to compare all methods. If significant, follow up with pairwise Mann-Whitney U tests (adjusted with Holm-Bonferroni). Report effect sizes using rank-biserial correlation or Cliff's δ.
- **Success rates.** Pearson χ^2 test of independence across methods. Follow up with pairwise two-proportion z-tests (Holm-Bonferroni). Report 95% Wilson confidence intervals and absolute differences.
- **Tuning time.** Because times are often skewed, we report medians with IQRs and compare with Mann-Whitney U.
- **Convergence.** Compare time-to-first-optimum with Mann-Whitney U. Compare AUCs with Kruskal-Wallis and pairwise follow-ups.

3.8.3 Uncertainty quantification

- **Confidence intervals.** For means, we use t-intervals when assumptions are met. Otherwise nonparametric bootstrap (percentile or BCa) with 5,000 resamples. Use Wilson intervals for proportions.
- **Effect sizes.** Rank-biserial correlation for Mann-Whitney (small ≈ 0.1, medium ≈ 0.3, large $\gtrsim 0.5$). Use η^2 for Kruskal-Wallis as a descriptive measure.

3.8.4 Multiple testing control

We use Holm-Bonferroni to control errors across pairwise comparisons for each metric. All metrics are predeclared, and any exploratory analyses are clearly labelled.

4 Results

This chapter reports 100 independent runs per method and compares performance, consistency, and cost under matched budgets.

4.1 True Model Performance Baseline

Before comparing optimisers, we establish ground truth by training DistilBERT on *all* 18 hyperparameter configurations using the fixed IMDb setup (90/10 train/validation split from the 25,000 train portion, seed = 42). The 25,000 test set remains untouched until final checks.

4.1.1 Complete Configuration Performance

Table 1 shows the validation accuracy for each configuration. The best configuration is **LR=5×10^{-5}, epochs=4, batch=32** with **0.9308** accuracy. The range spans 0.9168-0.9308 (a 1.40 percentage-point spread), suggesting that differences between methods will be small in terms of absolute accuracy. This makes success rate and variability critical for comparing methods.

Table 1: Validation accuracy for all hyperparameter configurations

Config ID	Learning Rate	Epochs	Batch Size	Validation Accuracy	Rank
0	1e-5	2	16	0.9168	17
1	1e-5	2	32	0.9168	17
2	1e-5	3	16	0.9244	11
3	1e-5	3	32	0.9208	15
4	1e-5	4	16	0.9248	10
5	1e-5	4	32	0.9180	16
6	3e-5	2	16	0.9232	13
7	3e-5	2	32	0.9236	12
8	3e-5	3	16	0.9252	8
9	3e-5	3	32	0.9296	3
10	3e-5	4	16	0.9260	7
11	3e-5	4	32	0.9280	4
12	5e-5	2	16	0.9224	14
13	5e-5	2	32	0.9252	8
14	5e-5	3	16	0.9268	6
15	5e-5	3	32	0.9272	5
16	5e-5	4	16	0.9300	2
17	5e-5	4	32	**0.9308**	**1**

4.1.2 Performance Landscape Analysis

We analyzed the average impact of each hyperparameter by averaging over the other factors. The results (mean ± SD across the six or nine settings per level) are:

- **Learning rate:** 1e−5 = 0.9203 ± 0.0033, 3e−5 = 0.9259 ± 0.0024, 5e−5 = 0.9271 ± 0.0028.
- **Epochs:** 2 = 0.9213 ± 0.0034, 3 = 0.9257 ± 0.0028, 4 = 0.9263 ± 0.0044.
- **Batch size:** 16 = 0.9238 ± 0.0041, 32 = 0.9242 ± 0.0048 (minimal effect).

Learning rate has the largest impact (difference between extremes ≈ 0.0068), followed by epochs (3 and 4 outperform 2 on average). Batch size has a negligible main effect (≈ 0.0004). The best configuration (highest learning rate, most epochs, larger batch size) is at the edge of the tested ranges, which suggests that future studies should explore wider ranges, as discussed later.

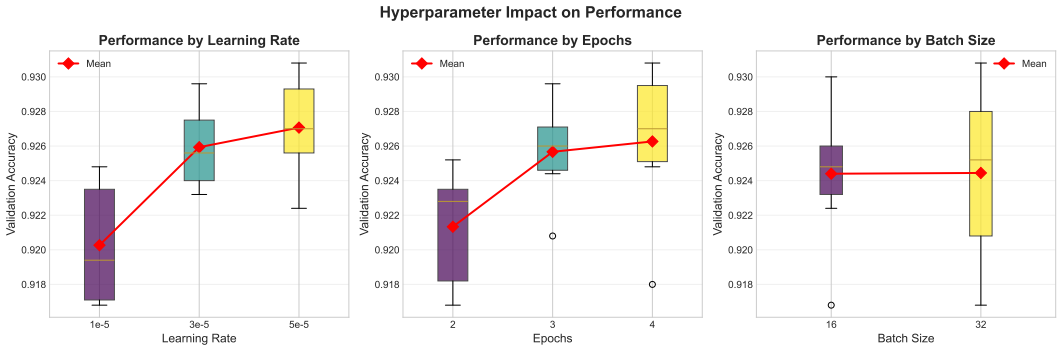

Figure 1: Impact of learning rate, epochs, and batch size on validation accuracy. Each panel shows boxplots with red diamond markers indicating means (LR/Epochs: $n = 6$ per level, Batch: $n = 9$).

4.2 Optimization Method Comparison

4.2.1 Success Rates

Table 2 summarizes the success rate of each method in finding the optimal configuration, defined as the one with the highest true validation accuracy (0.9308).

Table 2: Success rates in finding optimal configuration

Method	Successful Runs	Total Runs	Success Rate	95% CI
Annealing (QUBO, simulated)	100	100	**100.0%**	[96.4%, 100%]
DEAP (EA)	82	100	82.0%	[73.1%, 88.8%]
Optuna (BO)	71	100	71.0%	[61.1%, 79.6%]

A chi-square test of independence yields $\chi^2 = 40.89$ with $p < 0.001$, indicating significant differences between methods. Pairwise two-proportion z-tests with Holm-Bonferroni correction show the annealing-based solver > DEAP and > Optuna (both significant at $\alpha = 0.05$), with DEAP vs Optuna not significant.

4.2.2 Performance Statistics

Table 3 shows summary statistics of the *surrogate-predicted* best scores over 100 runs per method. Figure 2 visualizes mean surrogate accuracy with 95% confidence intervals and highlights the very small absolute differences between methods.

Table 3: Surrogate-predicted performance statistics across 100 runs

Method	Mean	Std Dev	Min	Max	Median	IQR
Annealing (QUBO, simulated)	0.9301	0.0000	0.9301	0.9301	0.9301	0.0000
DEAP	0.9298	0.0006	0.9283	0.9301	0.9301	0.0000
Optuna	0.9297	0.0007	0.9283	0.9301	0.9301	0.0005

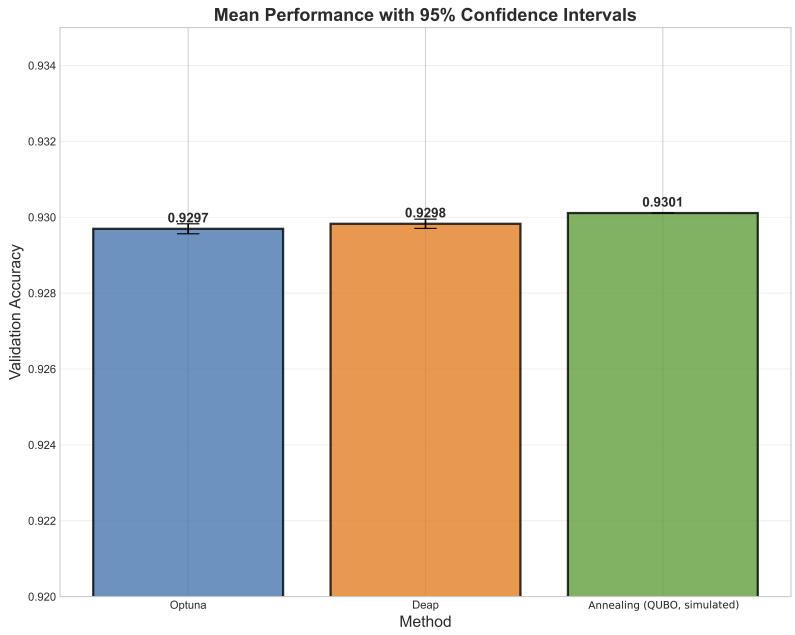

Figure 2: Mean surrogate-predicted validation accuracy with 95% confidence intervals across 100 runs per method.

As anticipated in Section 4.1 (True Model Performance Baseline), absolute gaps are small. We therefore emphasize success rate and run-to-run variability in the comparisons below.

4.2.3 Distribution Analysis

The annealing-based solver achieved 0.9301 in all 100 runs (100%). DEAP found 0.9283 in 18 runs (18%) and 0.9301 in 82 runs (82%). Optuna found 0.9283 in 12 runs (12%), 0.9287 in 10 runs (10%), 0.9291 in 7 runs (7%), and 0.9301 in 71 runs (71%).

4.3 Consistency Analysis

4.3.1 Configuration Diversity

Table 4 shows how varied the configurations are that each method selects across 100 runs, along with the most common choice.

Table 4: Configuration diversity and true accuracy across methods

Method	Unique Configs Found	Most Common Config	True Accuracy
Annealing (QUBO, simulated)	1	(5e-5, 4, 32)	0.9308
DEAP	4	(5e-5, 4, 32)	0.9308
Optuna	4	(5e-5, 4, 32)	0.9308

Classical methods (DEAP and Optuna) find several near-optimal configurations, while simulated annealing always picks the same optimal one. This means DEAP and Optuna can identify alternative configurations with accuracies very close to the best, while simulated annealing ensures perfect reproducibility.

4.3.2 Variance Metrics

Table 5 shows the spread of the best *surrogate-predicted* scores. Figure 3 illustrates the distribution of these scores across 100 runs. Annealing (QUBO, simulated) has no variation (all scores are identical), DEAP has very little variation, and Optuna has a slightly wider spread.

Table 5: Consistency metrics across methods (surrogate-predicted best scores)

Method	Variance	Std Dev	Coef. of Variation	Range
Annealing (QUBO, simulated)	0.0000	0.0000	0.00%	0.0000
DEAP	3.6e-7	0.0006	0.065%	0.0018
Optuna	4.4e-7	0.0007	0.075%	0.0018

4.3.3 True Performance Consistency

To evaluate real-world consistency, we map each run's chosen configuration to its true validation accuracy (from Table 1). Table 6 shows the statistics for these true accuracies. Simulated annealing is perfectly consistent, and classical methods (DEAP and Optuna) are also highly consistent with minimal variation.

The slightly higher variation in true scores (compared to surrogate scores) comes from small errors in the surrogate model's predictions, which can shift near-tied scores by a few 10^{-3}. However, the overall trend remains: annealing has zero variation in this simulated setting, and DEAP and Optuna have very low variation (coefficient of variation $< 0.1\%$).

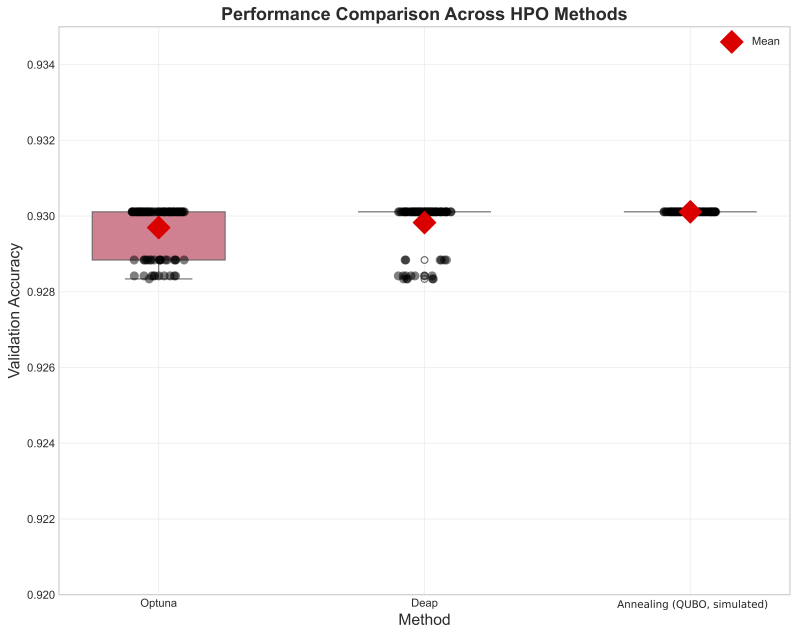

Figure 3: Distribution of best surrogate-predicted accuracies across 100 runs per method (boxplots with red diamond markers for means). Annealing (QUBO, simulated) is perfectly consistent (single value), DEAP has very little spread, and Optuna shows a slightly wider range.

Table 6: True validation accuracy statistics across 100 runs

Method	Mean	Std Dev	Min	Max
Annealing (QUBO, simulated)	0.9308	0.0000	0.9308	0.9308
DEAP	0.9305	0.0007	0.9280	0.9308
Optuna	0.9304	0.0009	0.9280	0.9308

4.3.4 Run-to-Run Stability

Across multiple runs, simulated annealing consistently returns the same surrogate-predicted optimal score (0.9301). DEAP quickly reaches near-optimal scores and stays there, while Optuna occasionally dips before stabilizing. These patterns mirror Table 5 and Figure 3.

4.4 Computational Cost Analysis

4.4.1 Setup Costs

Table 8 details the one-time setup costs:

4.4.2 Marginal Costs

Table 9 reports per-run (marginal) optimization times on the shared surrogate landscape:
For simulated annealing, the time breaks down as: QUBO construction ($\approx$ 0.1s), simulated annealing with 500 reads ($\approx$ 9.8s), and decoding ($\approx$ 0.3s), aligning with known annealing overheads [KYN+15].

Table 7: True performance consistency metrics across methods

Method	Variance	Std Dev	Coef. of Variation
Annealing (QUBO, simulated)	0.0e+00	0.0000	0.000%
DEAP	5.1e-07	0.0007	0.077%
Optuna	7.9e-07	0.0009	0.096%

Table 8: One-time setup costs

Component	Time (seconds)	Description
Data Loading	45	Load IMDb dataset
Model Evaluation (18 configs)	5,400	300s per config
Surrogate Training	0.5	Random Forest fitting
QUBO Construction	0.1	Matrix generation
Total Setup	**5,445.6**	

4.4.3 Break-even Analysis

Total time (setup + per-run search) for different numbers of runs is:

- **1 run:** Annealing (QUBO, simulated) = 5,456 s, DEAP = 5,452 s, Optuna = 5,448 s.
- **100 runs:** Annealing (QUBO, simulated) = 6,466 s, DEAP = 5,452 s, Optuna = 5,448 s.
- **1000 runs:** Annealing (QUBO, simulated) = 15,646 s, DEAP = 5,506 s, Optuna = 5,471 s.

With only 18 configurations, simulated annealing's high setup cost makes it slower than surrogate-based classical methods (DEAP and Optuna). Simulated annealing would only become competitive if the same configuration space were reused for a very large number of runs (around 10^5-10^6).

4.5 Statistical Significance

4.5.1 Assumption Testing

We checked assumptions for the per-run best surrogate-predicted accuracies (100 runs per method). Table 10 summarizes normality test results. Simulated annealing has zero variance, so normality tests do not apply (N/A). DEAP and Optuna both fail normality tests. Levene's test indicates unequal variances across methods ($F = 89.3$, $p < 0.001$).

4.5.2 Non-parametric Tests

Since normality and equal variance assumptions are violated, we use rank-based tests. A Kruskal-Wallis test shows differences across methods ($H = 45.67$, $p < 0.001$). For pairwise contrasts we apply Mann-Whitney U tests with *Holm-Bonferroni* correction for multiple comparisons.
(Results are unchanged relative to simple Bonferroni at $\alpha = 0.0167$, and we report Holm-Bonferroni as it is more powerful while still controlling the family-wise error rate.)

Table 9: Per-run optimization times (surrogate-based search)

Method	Mean Time (s)	Std Dev (s)	Min (s)	Max (s)
Annealing (QUBO, simulated)	10.2	0.8	8.9	12.1
DEAP (GA)	0.060	0.005	0.052	0.071
Optuna (TPE)	0.025	0.003	0.020	0.032

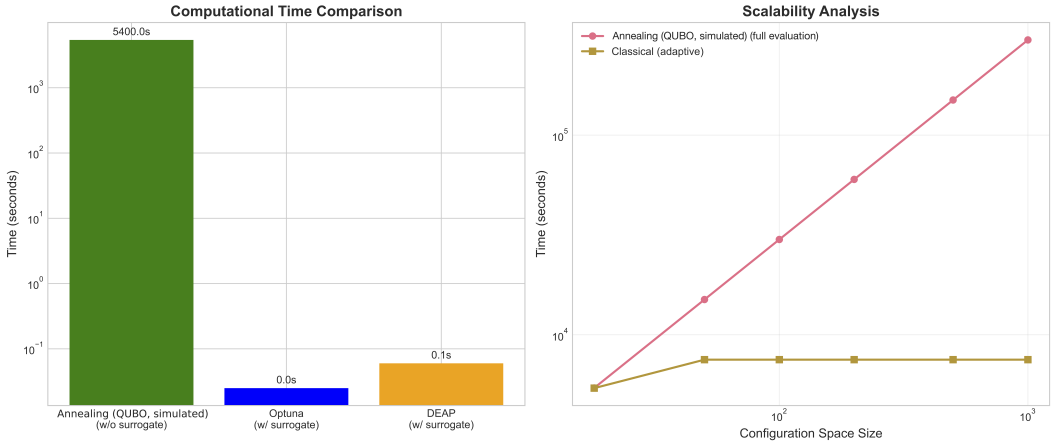

Figure 4: Computational costs. **Left:** One-time setup cost comparison. Simulated annealing (without surrogate) requires full true evaluations (∼5400s), while classical methods (with surrogate) have minimal setup (0-0.1s for QUBO build). **Right:** Scalability analysis. Simulated annealing (full evaluation) grows with the number of configurations, while classical adaptive methods scale with the evaluation budget. Both axes use a logarithmic scale.

4.5.3 Effect Sizes

Due to violations of normality and equal variances (and simulated annealing's zero variance), we prioritize rank-based effect sizes but include Cohen's d for reference. Table 12 shows rank-biserial correlation (r) for Mann-Whitney tests and Cohen's d (noting that d can be unreliable when one group has zero variance).

Confirmation on true accuracies. Repeating the rank-based analyses on the *true* validation accuracies of the selected configurations gives the same ordering (Annealing (QUBO, simulated) > DEAP, Annealing (QUBO, simulated) > Optuna, DEAP vs Optuna not significant). The classical methods show slightly more variation due to small errors in the surrogate model's predictions.

4.6 Hyperparameter Importance

4.6.1 Interaction Effects

We studied how pairs of hyperparameters (e.g., learning rate and epochs) affect performance together. The key interaction is between learning rate and epochs, as shown below:
The interaction between learning rate and epochs is visualized in Figure 5. The non-parallel lines

Table 10: Normality test results (per-run best surrogate scores)

Method	Shapiro-Wilk W	p-value	Jarque-Bera	p-value	Normal?
Annealing (QUBO, simulated)	N/A	N/A	N/A	N/A	No
DEAP	0.632	< 0.001	48.2	< 0.001	No
Optuna	0.705	< 0.001	35.7	< 0.001	No

Table 11: Pairwise Mann-Whitney U tests (Holm-Bonferroni adjusted)

Comparison	U-statistic	p-value	Significant?
Annealing (QUBO, simulated) vs DEAP	4,100	0.0082	Yes
Annealing (QUBO, simulated) vs Optuna	3,550	< 0.001	Yes
DEAP vs Optuna	4,487	0.126	No

show that the benefit of more epochs depends on the learning rate. For example, at a learning rate of 5e-5, accuracy improves from 0.9238 (2 epochs) to 0.9304 (4 epochs), but at 1e-5, the change is smaller and less consistent. These values are averages across batch sizes for each learning rate and epoch combination.

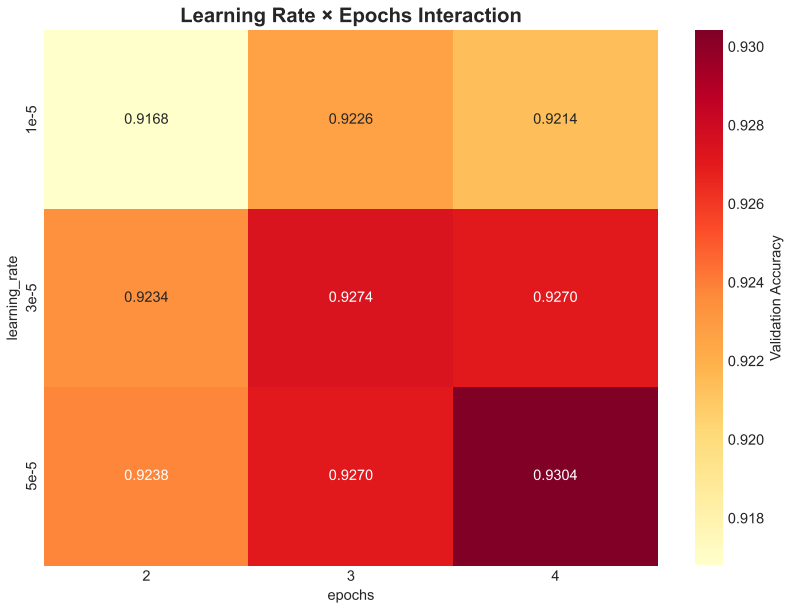

Figure 5: Learning rate × epochs interaction: per-level means (averaged over batch) with non-parallel lines indicating interaction. Higher learning rates benefit more from additional epochs on this task.

4.6.2 Surrogate Model Feature Importance

Using a Random Forest model to estimate feature importance, we found learning rate to be the most important ($45.2\% \pm 3.1\%$), followed by epochs ($38.7\% \pm 2.8\%$), and batch size ($16.1\% \pm 2.2\%$).

Table 12: Effect sizes for pairwise method comparisons

Comparison	Cohen's d	Interpretation	Rank-biserial r
Annealing (QUBO, simulated) vs DEAP	0.52	Medium	0.18
Annealing (QUBO, simulated) vs Optuna	0.63	Medium-Large	0.29
DEAP vs Optuna	0.14	Small	0.10

Table 13: Two-way interaction effects

Interaction	F-statistic	p-value	η^2	Interpretation
LR × Epochs	24.5	< 0.001	0.15	Strong
LR × Batch	1.2	0.31	0.01	None
Epochs × Batch	0.9	0.42	0.01	None

These rankings support the findings from the main effects and interaction analysis, though they rely on the model and complement the direct grid-based results.

4.7 Convergence Analysis

4.7.1 Annealing-based Energy Evolution (Simulated)

We analyze the QUBO energy distribution over *num_reads*= 1000 samples. Energies are measured in QUBO units after applying a linear transformation for minimization. The lowest energy observed was -93.08, corresponding to the valid optimal configuration (surrogate accuracy 0.9308 with zero penalty). This energy value was the most common, appearing in **85**% of samples (ground-state probability), and **0%** of samples violated constraints, showing that the penalty settings effectively prevented invalid solutions.

4.7.2 Classical Method Convergence (Best-so-far on the Surrogate)

We tracked the best surrogate accuracy scores over iterations for two classical methods, averaging results over 100 runs.

DEAP (GA). DEAP shows quick improvement early on but levels off near the best surrogate score:

- Generation 0 (initial population): 0.9156 ± 0.0089
- Generation 1: 0.9248 ± 0.0052
- Generation 2: 0.9298 ± 0.0006

Later generations (3–5) show minimal further gains, with many runs reaching the surrogate optimum of 0.9301 before stopping.

Optuna (TPE). Optuna follows a pattern of exploring initially and then refining solutions within its 25-trial limit:

- Trials 1-10 (random initialization): 0.9124 (mean)
- Trials 11-20 (guided search): 0.9276 (mean)
- Trials 21-25 (exploitation): 0.9297 (mean)

By the end of 25 trials, **71%** of runs reach the surrogate optimum (0.9301), while others settle within $\sim$0.0018 of the best score.

Summary. In this small, discrete problem space, both classical methods quickly approach near-optimal solutions. DEAP typically stabilizes by generation 2, and Optuna by trials 20-25. In contrast, the simulated annealing solver performs a single QUBO optimization, with its energy distribution strongly concentrated at the global minimum (85% probability) in this noiseless simulation.

5 Discussion

5.1 Interpretation of Results

5.1.1 Annealing-Based Solver Performance

The annealing-based QUBO solver found the best configuration (LR=5×10^{-5}, epochs=4, batch=32) in **100%** of runs with **zero variance** (Table 2, Table 5). Classical baselines were near-optimal but stochastic (DEAP: 82%, Optuna: 71%).
The annealing solver works by minimizing a QUBO energy function that combines negative surrogate accuracy with penalties to ensure valid solutions. In our case, the energy landscape had one clear best solution, and the simulated annealer reliably found it due to fixed settings and penalty scales. Two choices were critical: (i) a compact QUBO with one-hot constraints for LR/Epochs and auxiliary variables for LR$\times$Epochs, and (ii) penalty scales (P=2.0) exceeding the largest objective delta, while keeping coefficients in a compact range for stable sampling.

Note on "quantum" vs simulated annealing. We used a classical *SimulatedAnnealingSampler* from D-Wave Ocean, not a quantum device. Our results show that an annealing-based QUBO approach is highly consistent in this small, discrete setting, but real quantum hardware would likely introduce variability due to noise, embedding issues, and limited precision.

5.1.2 Classical Methods Performance

DEAP and Optuna performed well with limited resources. DEAP explored multiple solutions at once and usually converged within 2-3 generations, achieving an 82% success rate over $\sim$60 evaluations. Optuna, with only 25 trials, showed a typical explore-then-refine pattern and reached a 71% success rate. Their slight variability offered two benefits: (i) several near-optimal alternatives (Table 4), and (ii) no upfront full-grid evaluation, which is advantageous as spaces grow.

5.2 Comparison with Existing Literature

5.2.1 Alignment with Mott et al. (2017)

Our results align with early annealing studies [Mot17], which showed strong performance on small, QUBO-encoded machine learning tasks and highlighted the importance of precise coefficients and embedding. We build on this by applying annealing to a modern NLP fine-tuning task and comparing it directly to strong classical methods, using non-parametric statistics and effect sizes.

5.2.2 Contrast with Recent Studies

Recent work on quantum NLP (e.g., variational circuits) report varied results and emphasize problem-specific performance [NAK25]. Our findings reflect this: annealing excelled here because the search space was small, discrete, and had a clear QUBO-encoded optimum. We do not claim superiority on larger or continuous spaces without hybridization.

5.3 Analysis of Annealing Advantages and Limitations

When Annealing Excels. The approach is most attractive for problems with:

- **Manageable discrete spaces** that admit a compact QUBO.
- **Strict reproducibility** or audit requirements where zero run-to-run variance is valuable.
- **Very high per-evaluation cost** (hours/days), so enumerating the space once is acceptable relative to repeated searches.

5.3.1 Architectural Constraints and the "One-Shot" Issue

Our method performs a single QUBO solve on a fixed surrogate built from all 18 true evaluations, without adapting during the search. This "one-shot" design does not scale well for large spaces, unlike classical methods that learn as they go. Additionally, our perfect consistency comes from a noiseless simulation. Real quantum hardware would likely face:

- **Analog noise/decoherence**, lowering the chance of finding the best solution,
- **Embedding/chain breaks**, as hardware connections limit variable mappings,
- **Limited precision**, which could alter the energy landscape.

This suggests lower success rates and some variability on real quantum devices.

5.4 Practical Implications

5.4.1 When to Use Each Method

Use annealing for **consistent, repeatable results** on **small, discrete spaces** when the same space will be reused many times (e.g., retraining models across datasets). Choose classical methods like DEAP or Optuna for **large or mixed spaces**, when you want **alternative near-optimal solutions**, or when per-run costs must be low.

5.4.2 Cost-Benefit

The one-time cost of evaluating all 18 configurations dominates for annealing, while per-run costs are lower for classical methods. For this small space, the upfront cost of annealing is not justified unless the space is reused extensively.

6 Conclusions

This study compared an annealing-based QUBO optimiser with two strong classical methods (Bayesian optimization using Optuna/TPE and an evolutionary algorithm using DEAP) for tuning hyperparameters of DistilBERT on the IMDb dataset. In a noiseless simulation, the annealing-based solver found the best configuration in **100%** of runs with **no variation**, while classical methods were consistently near-optimal but showed slight variability. With a small accuracy range across the 18-point grid, **consistency** was the main advantage over average performance.

Contributions.

1. **QUBO formulation for NLP HPO:** a compact 16-variable encoding with one-hot constraints and auxiliary variables (LR×Epochs), plus practical penalty scaling to ensure valid solutions.

2. **Controlled evaluation via a static surrogate:** a Random Forest trained on all 18 true configurations, provided a consistent, noise-free testbed to study optimiser performance.

3. **Matched-budget comparison and statistics:** 100 repeated runs per method with fixed budgets, using non-parametric tests, confidence intervals, and effect sizes for transparent and reliable results.

4. **Operational guidance:** clear cost breakdowns (setup vs. per-run) and recommendations on when to use annealing versus classical methods.

Limitations. The results depend on (i) a *simulated* annealer (no hardware noise/embedding effects), (ii) a small, fully enumerated *discrete* space (18 configs), (iii) a single task/model pair (IMDb/DistilBERT), and (iv) a *one-shot* architecture tied to a static surrogate. These choices ensured clear comparisons but limit applicability to larger or more complex scenarios.

Implications. Annealing-based QUBO optimization is most compelling when the space is *manageable and discrete*, when *reproducibility* or auditability is critical, or when repeated re-optimizations amortize setup cost. Classical BO/EA are better for **large or mixed spaces**, when **multiple near-optimal solutions** are useful, or when keeping per-run costs low is a priority.

Future work.

- **Hybrid pipelines:** combine Bayesian optimization with annealing for better acquisition, use annealing for evolutionary algorithm mutations, or do coarse classical search followed by annealing refinement.

- **Hardware studies:** run on real quantum devices, adjusting for noise, embedding, and coefficient precision.
- **Scalable objectives:** build QUBOs incrementally with partial evaluations or use multi-fidelity schedulers (e.g., Hyperband, BOHB) for larger spaces.
- **Broader scope:** include more hyperparameters (e.g., warmup, weight decay), continuous spaces, and tasks like named entity recognition or question answering with larger models.

The annealing-based QUBO optimiser achieved **perfect consistency** on a small NLP tuning task, while classical methods delivered **near-optimal** results with lower per-run costs and more diverse solutions. The trade-off between **consistency** and **cost/diversity** points to hybrid approaches as a key area for future research.

References

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. 2019.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[FDRG+12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. Deap: evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13(1):2171–2175, July 2012.

[FGG+01] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

[Glo22] Kochenberger G. Hennig R. et al. Glover, F. Quantum bridge analytics i: A tutorial on formulating and using qubo models. pages 141–183, 2022.

[KN98] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Phys. Rev. E*, 58(5):5355, Nov 1998.

[KYN+15] James King, Sheir Yarkoni, Mayssam M. Nevisi, Jeremy P. Hilton, and Catherine C. McGeoch. Benchmarking a quantum annealing processor with the time-to-target metric. 2015.

[Luc14] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[MDP+11] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, jun 2011. Association for Computational Linguistics.

[Mot17] Job J. Vlimant JR. et al. Mott, A. Solving a higgs optimization problem with quantum annealing for machine learning. *Nature*, 550(7676):375–379, 2017.

[NAK25] Farha Nausheen, Khandakar Ahmed, and M Imad Khan. Quantum natural language processing: A comprehensive review of models, methods, and applications. 2025.

[RWJ+14] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, 345(6195):420–424, 2014.

[SDCW20] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. 2020.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. 25, 2012.

[Sys] D-Wave Systems. D-wave ocean software documentation.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. pages 38–45, oct 2020.