

# Investigating the performance of Kolmogorov-Arnold networks in addressing optimisation problems

Name: Arjun Parmar

ID: F121572

Supervisor: Alia Asheralieva

Department of Computer Science

Loughborough University

# Abstract

Kolmogorov-Arnold networks (KANs) represent a new approach to neural network architecture, leveraging B-spline activation functions on edges rather than traditional node-based activation functions, which allows for greater adaptability and interpretability. Optimisation plays a critical role across many real-world domains, from logistics and finance to engineering design, and neural network approaches have proven to be effective at solving some of the most complex examples of these. This project explores the efficacy of KANs in addressing mathematical optimisation problems, including linear, integer, non-convex, and stochastic optimisation. By comparing KANs to traditional multi-layer perceptrons (MLPs) implemented using libraries such as PyTorch, the relative strengths and weaknesses of these architectures in solving complex optimisation tasks can be evaluated. Results demonstrate that KANs generally show superior performance in terms of accuracy, convergence speed, and stability, particularly in non-convex and stochastic settings, while also maintaining good performance on linear and integer problems. Although KANs require significantly higher computation time, they achieve near-optimal solutions in significantly fewer training iterations. This project provides a detailed performance analysis of KANs and highlights their potential advantages and limitations in broader optimisation contexts. The Pykan library is utilised to provide an implementation of KANs.

# Keywords

Machine learning, Multi-layer perceptrons, Kolmogorov-Arnold networks, Optimisation, Linear optimisation, Integer optimisation, Non-convex optimisation, Stochastic optimisation

# Contents

1. Introduction.....	5
1.1. Motivation.....	5
1.2. Current research and gaps .....	5
1.3. Objectives.....	6
1.4. Organisation of thesis.....	6
2. Literature Review .....	7
2.1. Multi-Layer Perceptrons (MLPs).....	7
2.2. Kolmogorov-Arnold Networks (KANs) .....	8
2.3. Optimisation Problems.....	11
2.4. Analysis of Neural Networks in Optimisation Tasks.....	13
2.4.1. Linear Optimisation.....	13
2.4.2. Integer Optimisation.....	13
2.4.3. Non-Convex Optimisation .....	14
2.4.4. Stochastic Optimisation.....	14
3. Methodology .....	16
4. Linear Optimisation .....	18
4.1. Problem 1 – Lower Dimensional .....	18
4.1.1. Description .....	18
4.1.2. Analysis .....	20
4.2. Problem 2 – Higher Dimensional.....	27
4.2.1. Description .....	27
4.2.2. Analysis .....	28
4.3. Cross-problem Comparison .....	34
5. Integer Optimisation .....	35
5.1. Problem 1 – Lower Dimensional .....	35
5.1.1. Description .....	35
5.1.2. Analysis .....	37
5.2. Problem 2 – Higher Dimensional.....	42
5.2.1. Description .....	42

5.2.2. Analysis .....	44
5.3. Cross-problem Comparison .....	49
6. Non-Convex Optimisation.....	50
6.1. Problem 1 – Lower Dimensional .....	50
6.1.1. Description .....	50
6.1.2. Analysis .....	51
6.2. Problem 2 – 3D Rastrigin function .....	57
6.2.1. Description .....	57
6.2.2. Analysis .....	59
6.3. Cross-problem Comparison .....	64
7. Stochastic Optimisation .....	64
7.1. Problem 1 – Lyapunov Optimisation .....	65
7.1.1. Description .....	65
7.1.2. Analysis .....	67
7.2. Problem 2 – Budget Constrained Price Optimisation .....	72
7.2.1. Description .....	72
7.2.2. Analysis .....	73
7.3. Cross-problem Comparison .....	77
8. Discussion.....	78
9. Conclusion .....	80
10. Bibliography .....	81
11. Appendices.....	82

# Introduction

## Motivation

Optimisation problems are a fundamental part of many real-world applications. Many industries rely on solving mathematical optimisation problems to make better decisions. A simple example is determining the optimal mix of products in a factory to maximise profits, taking into consideration constraints such as time and resources needed to manufacture those products. Another ubiquitous example is that of financial modelling, being able to optimise the stock portfolios of customers according to the constraints set by the market.

These problems can be split into maximisation and minimisation problems, which respectively involve maximising or minimising the value of an objective function according to some constraints. These problems can take many forms, such as linear, integer, non-convex, and stochastic, which are the classes of problems considered in this project.

Classical mathematical methods exist to solve many optimisation problems, but they are often not as practical for more complex cases, especially when dealing with non-convex or high-dimensional spaces. Neural network approaches, particularly those involving multi-layer perceptrons, have shown promise in solving these types of problems by approximating highly accurate solutions instead of solving them mathematically. However, they still may suffer from disadvantages such as overfitting, slow convergence, and difficulty in handling constraints.

Recently, a new type of neural network architecture has been proposed, called Kolmogorov-Arnold networks. These utilise a B-spline activation function on the edges of the network, rather than activation functions on the nodes, and this change is hypothesised to lead to better performance on challenging optimisation tasks. This report will examine the efficacy of KANs in solving various types of optimisation problems, using code to compare and analyse KAN approaches to traditional MLPs, in the hope of understanding where KANs may offer genuine advantages and where their limitations lie. Advances in this area could enable more efficient, scalable optimisation strategies across a wide range of practical domains.

## Current research and gaps

Currently there is very little to no research on applying KANs to optimisation problems, making this a novel approach. However, what has been shown is their ability to efficiently represent complex mathematical functions with fewer parameters and enhanced interpretability compared to deep learning models [1]. Promising results in symbolic regression, physics-informed machine learning, and function decomposition [2], show that areas requiring high precision and mathematical structure could benefit from KANs.

While KANs excel at approximating continuous functions, their effectiveness in structured optimisation tasks remains unknown, such as in problems with discrete constraints, non-convex

objectives, or stochastic dynamics. Existing work does not systematically compare KANs to standard architectures like MLPs, benchmarking them across metrics such as convergence speed, accuracy, and computational cost. This project addresses these gaps by applying KANs to a wide range of optimisation problems, evaluating their performance systematically against MLPs, and providing an assessment of their strengths and limitations in mathematical optimisation.

## Objectives

- Investigate the effectiveness of KANs in solving optimisation problems, including linear, integer, non-convex, and stochastic problems.
- Compare KAN-based approaches with traditional MLP based approaches.
- Assess the performance of KANs in dynamic, uncertain settings through stochastic optimisation problems involving real time constrained decision-making.
- Analyse the performance of KANs in terms of metrics such as accuracy, convergence speed, and computational efficiency.
- Analyse strengths and limitations of KANs when used in optimisation and suggest areas for future research.

## Organisation of thesis

The thesis has been structured to explore the capabilities of KANs in solving a range of optimisation problems, while providing a comparison to an MLP approach. In the introduction and literature review, background information and context are established, followed by a methodology section which details the experimental setup. The main body of the thesis is divided into four sections, one for each specific optimisation class - linear, integer, non-convex, and stochastic - with two problems explored within each. For every problem, both a description and performance analysis are provided, as well as a comparison of the results from each problem. The thesis concludes with a discussion of the overall findings and suggestions for future research. The appendices section includes a link to the notebooks containing the relevant code and supporting results.

# Literature Review

This project's aim is to investigate the efficacy of KANs in solving optimisation problems. This is a largely unexplored area due to the fact that KANs are only a recent development. However, various neural networks of different approaches have been used to solve many optimisation problems, many with success, which provides a baseline from which comparisons can be made.

In this literature review, an examination will be conducted of the different types of neural networks and optimisation problems available to us, approaches that have been taken in the past, and critically analyse how these insights could inform our research.

## Multi-Layer Perceptrons (MLPs)

MLPs are a fundamental type of artificial neural network, commonly used for solving optimisation problems. MLPs consist of interconnected nodes with weights and biases, applying activation functions to simulate complex mappings between inputs and outputs. They are based on the universal approximation theorem (UAT) [3], which states that feedforward neural network with a single hidden layer can approximate any continuous function on a compact subset of  $R^n$  to any desired degree of accuracy. Formally, if  $\mathcal{C}(K)$  is a compact space on a set  $K \subseteq R^n$  where continuous functions are defined, then for any continuous function  $f \in \mathcal{C}(K)$  and any  $\varepsilon > 0$ , there exists a feedforward neural network  $\hat{f}$  with a single hidden layer such that:

$$|f(x) - \hat{f}(x)| < \varepsilon \quad \text{for all } x \in K \quad (2.1)$$

This essentially guarantees that neural networks are able to represent highly complex functions, regardless of how intricate the mapping between inputs and outputs is. This makes them highly versatile, allowing them to address a variety of mathematical problems, including linear and non-linear optimisation problems. However, this does not guarantee that the approximation is easy nor efficient to compute. The challenges MLPs can face include [4]:

- **High computational demands** - A large amount of training data and computational resources are required to achieve high accuracy.
- **The Vanishing Gradient problem** - This is where the gradients used to update a network during the backpropagation process become extremely small.
- **Local minima** - converging to sub-optimal solutions, especially for non-convex problems.
- **Overfitting** - especially when dealing with small data sets or complex network architectures.

However, MLPs remain the default choice due to their simplicity and adaptability, and consequently will serve as a critical benchmark for evaluating the performance of alternative neural network architectures like KANs.

## Kolmogorov-Arnold Networks (KANs)

KANs are a relatively new neural network architecture inspired by the Kolmogorov-Arnold representation theorem (KART) [5], which states that any continuous multivariate function can be expressed as a composition of univariate functions and sums. Unlike traditional MLPs, which use weights and activation functions on nodes, KANs utilise learnable B-spline activation functions on the edges, which enable a more efficient and interpretable representation of functions. Any continuous function  $f: [0,1]^n \rightarrow \mathbb{R}$  can be written as:

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \phi_q \left( \sum_{j=1}^m \psi_{qj}(x_j) \right) \quad (2.2)$$

where  $\phi_q$  and  $\psi_{qj}$  are continuous univariate functions. KART guarantees an exact representation of a multivariate function, unlike the UAT which only asserts that neural networks can approximate any function. This essentially provides a blueprint for an architecture that combines simpler functions to form a complex mapping. However, the problem is that the univariate functions can be non-smooth and highly irregular, meaning that it may not be efficient to compute or learn these functions. Hence, directly using this theory can be impractical. In Liu's paper on KANs [1], a way to overcome this limitation is described.

### Explaining basis functions

Typical activation functions on nodes include ReLU and Sigmoid, and a function can be represented by stacking many layers of these. With KANs, these are replaced by B-spline basis functions on edges, and the nodes just represent a summation step. B-splines are smooth, piecewise polynomial functions defined over intervals called knots. By combining many adaptable curves, KANs are able to represent complex functions. Each basis function represents a small part of the input, meaning that the model is more flexible and resistant to overfitting. Hence, by using a relatively small number of parameters, highly complex functions can be represented.



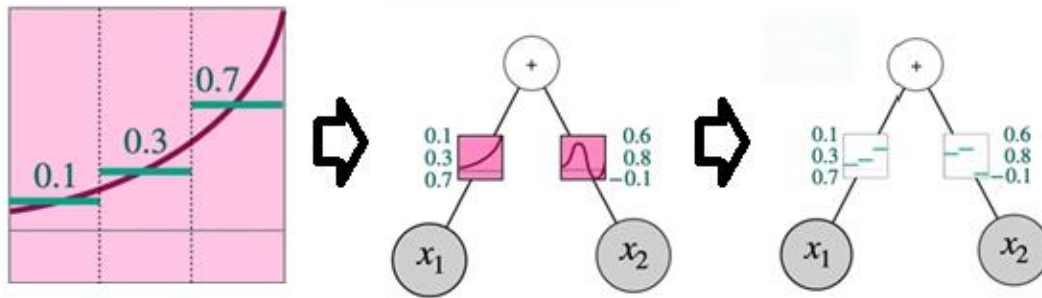


Figure 4-1: Simplified representation of how KAN basis functions work. [6]

The grid divides the input domain into intervals. Within each interval, a basis function is defined, and the network learns coefficients for these functions during training. In the above example, the curve is sectioned into 3 grids, and each is associated with the average value of the curve within. This gives a fairly accurate approximation of the curve, with 3 trainable parameters. In this example,  $[0.1, 0.3, 0.7]$  gives the knot vector, which is what the KAN will use to represent this curve. This allows the KAN to approximate functions with varying degrees of smoothness and complexity by adjusting the shape and placement of the functions. As training progresses, the grid can be refined further to allow modelling finer details, which focuses computational resources on areas more likely to be optimal.

These basis functions are adaptive, meaning they can be adjusted during training. They are also local, meaning changes in one part of the input space do not drastically affect others. This local control is what gives KANs their advantage in modelling detailed structures in optimisation landscapes.

KANs have already shown success in applications ranging from function approximation, classification, and regression. They have particularly excelled in scenarios requiring high accuracy with fewer parameters [1]. Their local adaptation through B-spline activation functions enhances computational efficiency and limits overfitting. However, challenges exist, such as computational expense in high-dimensional inputs and sensitivity to noisy data, limiting their scalability in some contexts [7].

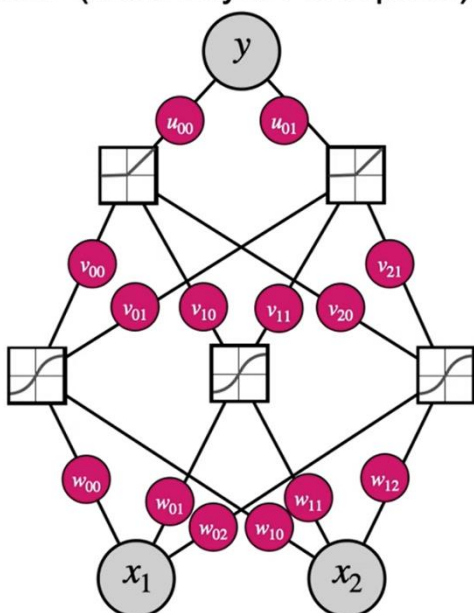
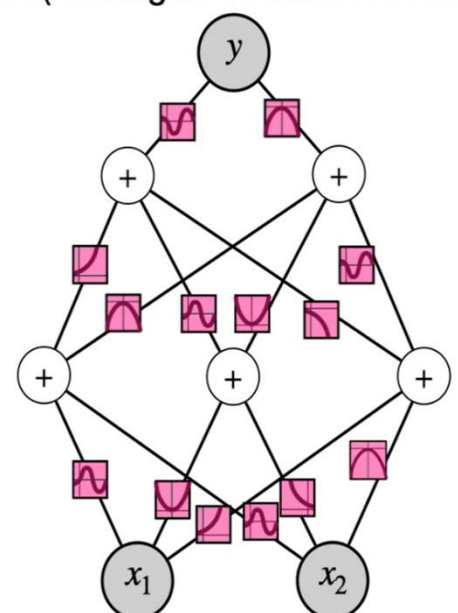
These properties suggest that KANs could excel at solving mathematical optimisation problems, given the need for precise representation of constraints and objectives.

The advantages that KANs gain can be summarised as follows:

- **Increased efficiency** - KANs can represent optimisation landscapes with fewer parameters, potentially converging faster and using less computational power.
- **Better generalisation** - Their compact structure helps reduce overfitting and enables effective learning even in high-dimensional or dynamic problems.
- **Improved interpretability** - The spline-based formulation allows for a clearer understanding of how inputs map to outputs, which provides greater model transparency.

- **Superior handling of non-linearity** - Their smooth, flexible activation functions are a natural advantage in approximating non-linear optimisation surfaces, potentially outperforming MLPs in non-convex and stochastic optimisation tasks.

Despite these advantages, KANs do face some drawbacks. For instance, the computational overhead of evaluating spline functions and their parameters at every step could offset efficiency gains that KANs offer. Furthermore, typical MLP algorithms benefit from parallelisation by applying the same activation function uniformly across nodes, whereas KANs have a unique activation function on each edge, introducing more computational costs [1]. Additionally, being in a relatively early stage of research on KANs means there is limited available evidence on their performance in various optimisation problems, making this a promising area for investigation.

	MLP	KAN
<b>Structure</b>	<p><b>MLP (Multi-Layer Perceptron)</b></p>  <p>Fixed activation functions Train weights</p> <p>[6]</p>	<p><b>KAN (Kolmogorov-Arnold Network)</b></p>  <p>Fixed weights Train activation functions</p>
	Fully connected layers with activations applied to nodes, using functions such as ReLU and sigmoid.	Grid-based spline functions applied to edges between layers.
<b>Theorems</b>	UAT - any continuous function can be approximated by a feedforward neural network with a single hidden layer and sufficient width. [3]	KART - any multivariate function can be expressed using compositions and sums of univariate functions. [5]

<b>Pros</b>	Widely used and understood. simple to implement. scalable. [8]	Local learning capability. better interpretability. efficient for low-dimensional problems. grid refinement enables adaptive focus. [1]
<b>Cons</b>	Can require many parameters. limited interpretability. struggles with constraint enforcement. [8] [9]	More complex to implement. higher initial training time. sensitive to hyperparameter tuning. [1]

Table 4-1: Summary of comparison between MLPs and KANs.

## Optimisation Problems

Optimisation problems are central to numerous fields, including mathematics, engineering, economics, and computer science. These problems involve finding the best solution from a set of feasible solutions, often subject to constraints. Traditional mathematical methods exist for solving these, which may be highly efficient and reliable for certain well-structured problems. However, their performance may degrade with increasing complexity, dimensionality, or the presence of non-convexity and uncertainty, hence leading to the idea of using neural networks to solve them.

Optimisation problems can broadly be classified into several categories:

- **Linear programming (LP):** maximising or minimising a linear objective function according to linear constraints. These are typically solved using the simplex or interior point methods [10]. These problems take the form of:

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t. : } & Ax \leq b, x \in R^n \end{aligned} \tag{2.3}$$

Where:

$x \in R^n$ : the decision vector (n continuous variables).

$c \in R^n$ : the cost vector, defining the weights in the objective.

$c^T x$ : the dot product, representing the total objective value to minimise.

$A \in R^{m \times n}$ : the constraint matrix, encoding mmm linear inequality constraints.

$b \in R^m$ : the right-hand-side constraint vector.

$Ax \leq b$ : the set of linear inequality constraints the solution must satisfy.

- **Integer programming (IP):** where the variables are constrained to integer values. If a mixture of integer and non-integer constraints are allowed, then it is called a mixed integer optimisation problem. These problems are highly applicable to real life situations, such as logistics and resource allocation, but are also more difficult to solve due to the integer constraints, being NP-hard. NP-hard problems are a class of problems for which no known algorithm can solve all instances efficiently (in polynomial time) [11]. These problems take the form of:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s. t. : } & Ax \leq b, x \in \mathbb{Z}^n \end{aligned} \quad (2.4)$$

Where the only differing term to the LP definition is:

$x \in \mathbb{Z}^n$ : the decision vector, with n integer-valued variables.

- **Non-convex optimisation:** as opposed to convex optimisation, where the solution is a single optimum point, non-convex optimisation focuses on cases where there are multiple local optima, making the search space more difficult. This involves making the objective function or constraints non-convex. [10]

$$\begin{aligned} \min \quad & f(x) \\ \text{s. t. : } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_j(x) = 0, j = 1, \dots, p \end{aligned} \quad (2.5)$$

Where:

$x \in \mathbb{R}^n$ : the decision vector (n continuous variables).

$f(x)$ : the non-convex objective function to minimise.

$g_i(x) \leq 0$ : the set of m inequality constraints.

$h_j(x) = 0$ : the set of p equality constraints.

- **Stochastic optimisation:** where uncertainty is added to an optimisation problem by using a probability distribution to change aspects of the problem. These are used in financial planning, supply chain management, and energy systems. [12]

$$\begin{aligned} \min \quad & E_\xi[f(x, \xi)] \\ \text{s. t. : } & x \in \mathcal{X} \end{aligned} \quad (2.6)$$

Where:

$x \in \mathcal{X}$ : the decision vector constrained to the feasible set  $\mathcal{X}$ .

$\xi$ : the random variable representing uncertainty or stochastic elements in the problem.

$f(x, \xi)$ : the objective function depending on both the decision variables and the random outcome.

$E_\xi[f(x, \xi)]$ : the expected value of the objective function over the distribution of  $\xi$ , representing the average performance across uncertainties.

## Analysis of Neural Networks in Optimisation Tasks

MLPs will serve as a baseline for assessing the efficacy of KANs at solving optimisation problems. For each problem domain, we will examine the effectiveness of various neural network approaches, including any shortcomings that could be overcome by KANs.

### Linear Optimisation

LP problems have been widely studied, and established mathematical methods exist for solving them. The Simplex method is one of the most effective approaches, providing exact solutions. It is particularly effective for solving smaller scale problems, since its worst-case time complexity is exponential. However, studies have shown it has satisfactory performance in most real-world cases [13]. Another widely used mathematical method is the Interior Point method, which has been shown to be more effective for solving larger scale LP problems [14] due to its polynomial time complexity.

MLPs also have been shown to be effective at solving LP problems [15]. Instead of using traditional formulae to get an exact solution, they can approximate a solution by training, allowing them to get a good solution quickly. This enables them to handle problems with higher dimensions faster. However, they are not able to get exact solutions like the mathematical methods mentioned above, and extensive tuning of the network is required.

### Integer Optimisation

Integer programming involves optimisation problems where some or all variables are constrained to take integer values, making it a particularly challenging class of problems as they are NP-hard. Traditional mathematical methods like Branch-and-Bound and Branch-and-Cut are widely used [16], systematically dividing the solution space into smaller subproblems to manage the combinatorial complexity of integer constraints. These methods are precise, but they are less efficient for large-scale applications because of the exponential increase in computational cost as the problem size increases, especially with high-dimensional or mixed-integer constraints.

Neural network-based approaches, such as MLPs and Long Short-Term Memory (LSTM) networks, have been explored as alternatives for solving IP problems. Neural networks excel in handling high-dimensional and dynamic environments by leveraging their ability to learn patterns and approximate solutions efficiently. For example, LSTM frameworks have shown notable improvements in solving mixed-integer programs, outperforming traditional MLPs in scalability and computational efficiency by a factor of 9 [17]. These models can approximate solutions rapidly, making them particularly valuable in scenarios requiring real-time decision-making or with large search spaces. However, because they provide approximations of solutions, they lack the precision of the mathematical techniques mentioned. Neural network approaches excel in cases involving uncertainty, dynamic environments, or high-dimensional problems.

## Non-Convex Optimisation

Non-convex optimisation presents significant challenges due to the existence of multiple local minima, making optimisation highly nontrivial, compared to the case of convex optimisation where there is only one minimum. Mathematical methods such as Sequential Quadratic Programming (SQP) and Gradient Descent are frequently used [18]. Even so, they often suffer from convergence to local minima unless enhanced by techniques such as random restarts.

MLPs and other neural networks are increasingly used to approximate solutions for non-convex optimisation problems, utilising their ability to model more complex, non-linear relationships. Despite their potential, typical neural networks face challenges such as the vanishing gradient problem during training, making it difficult to navigate highly non-convex landscapes [19]. Hybrid approaches combining reinforcement learning or genetic algorithms have shown promise for particular non-convex problems, trading off precision for scalability and adaptability. KANs are predicted to be significantly more effective than current MLP approaches due to their advantages in limiting the vanishing gradient problem.

## Stochastic Optimisation

Stochastic programming deals with optimisation under uncertainty, requiring methods that incorporate inputs or constraints based on a probability distribution. Mathematical approaches include the Sample Average Approximation (SAA) and Stochastic Gradient Descent (SGD). SAA is computationally intensive but effective for scenarios where uncertainty can be sampled extensively [12]. On the other hand, SGD is faster and better suited for large-scale problems, but due to stochastic noise may converge to suboptimal solutions.

Neural networks, including MLPs and Recurrent Neural Networks (RNNs), have been applied to stochastic problems, particularly in dynamic settings requiring adaptability over time. RNNs have been shown to be effective in time-series prediction tasks, which often underpin stochastic optimisation scenarios [20]. However, these approaches struggle with interpretability and can require significant computational resources for training. Recent work combining neural networks with probabilistic programming frameworks has shown potential for improving adaptability and better handling uncertainty in a stochastic environment, making them a promising area of research [21]. KANs, with their advantages in interpretability, could offer a more effective alternative.

Problem Type	Classical Methods	Neural Approaches	KAN Insights
<b>Linear Programming (LP)</b>	<ul style="list-style-type: none"> <li>- Simplex</li> <li>- Interior Point (polynomial) [13] [14]</li> </ul>	<ul style="list-style-type: none"> <li>- Can approximate solutions quickly</li> <li>- Requires tuning and does not reach exact solutions [15]</li> </ul>	<ul style="list-style-type: none"> <li>- Efficient with fewer parameters</li> <li>- Could outperform MLP in stability and convergence</li> </ul>
<b>Integer Programming (IP)</b>	<ul style="list-style-type: none"> <li>- Branch and bound</li> <li>- Precise but scale poorly (NP-hard) [16]</li> </ul>	<ul style="list-style-type: none"> <li>- MLPs and LSTMs used for high-dimensional, dynamic problems</li> <li>- Fast approximations but less accurate [17]</li> </ul>	<ul style="list-style-type: none"> <li>- Handles structured discrete patterns</li> <li>- Promising for scalable approximations</li> </ul>
<b>Non-Convex Optimisation</b>	<ul style="list-style-type: none"> <li>- SQP, Gradient Descent</li> <li>- Prone to local minima [18]</li> </ul>	<ul style="list-style-type: none"> <li>- Can model complex landscapes</li> <li>- Suffers from vanishing gradients [19]</li> </ul>	<ul style="list-style-type: none"> <li>- Reduces vanishing gradient issues</li> <li>- Learns sharper transitions; more stable</li> </ul>
<b>Stochastic Optimisation</b>	<ul style="list-style-type: none"> <li>- Sample Average Approximation (accurate, costly)</li> <li>- SGD (fast, noisy) [12]</li> </ul>	<ul style="list-style-type: none"> <li>- MLPs and RNNs adapt to dynamic, uncertain inputs</li> <li>- High computational cost and low interpretability [20] [21]</li> </ul>	<ul style="list-style-type: none"> <li>- Combines adaptability with structure</li> <li>- Interpretable and efficient under uncertainty</li> </ul>

Table 4-2: Summary of optimisation problems and insights into how they could be solved.

# Methodology

The general approach taken is to consider 2 problems within each category of optimisation problem, ranging from easier to more difficult. For each problem, an attempt will be made to solve it using an MLP based model, and another attempt will be made using a KAN based model. Analysis will be conducted to see which approach is better, based on metrics defined below. The methodology follows a consistent, and repeatable pipeline across all problem types to ensure fairness and comparability.

To ensure systematic evaluation, the following structure was applied across all problem types:

1. **Problem formulation** - each optimisation problem is formally defined with its objective function and constraints. Where possible, ground truth or analytical solutions are derived to serve as benchmarks. The 2 problems within each category range from lower dimensional to higher dimensional.
2. **Data generation and constraint handling** - for each problem, synthetic data is generated that obeys the respective problem constraints. In continuous optimisation, feasible input vectors are sampled randomly within the domain, then filtered using constraint functions. For integer optimisation, discrete behaviour is enforced by using only integer samples. In stochastic settings, data is generated dynamically (such as through Poisson arrivals or random budget distributions), and multiple runs are performed to capture uncertainty and assess stability.
3. **Model architectures** - two neural network architectures are implemented for each problem:
  - **MLP**: a conventional feedforward neural network using ReLU activations, constructed using the PyTorch library, trained with a fixed number of layers and neurons to try and match the complexity of the KAN.
  - **KAN**: constructed using the Pykan library [22]. The architecture, grid refinement and polynomial degree ( $k$ ) are tuned based on the difficulty of the problem.
4. **Training procedures** - both models are trained using Adam or AdamW optimisers for a fixed number of epochs (typically 1000–20000, depending on the problem). Adam was used in cases where simple adaptive learning sufficed, while AdamW was preferred for models requiring more robust generalisation. AdamW decouples the weight decay step, applying it directly to the model parameters, unlike Adam. This leads to more stable training and improved generalisation. [23]

Loss functions incorporate penalties where necessary to enforce feasibility, for example soft constraint penalties for budget violations. Convergence is monitored during training to capture model behaviour. For stochastic problems, data generation and model



evaluation are repeated across multiple random seeds to ensure robustness. Baseline solutions are computed analytically or using optimisation solvers.

5. Performance is evaluated using a consistent set of metrics:

- **Objective value** - the final value of the optimisation objective achieved by the model.
- **Final loss** - the final value of the loss function after training. The loss functions vary per problem, with most using a mean squared error-based loss with other penalties included.
- **L2 norm** – Euclidean distance of the model’s prediction to the actual solution. This provides a view of how the model’s predictions change over epochs.
- **Absolute error** – the absolute difference of the model’s objective value to the actual objective value.
- **Convergence time** - the number of epochs necessary for the model converge to a stationary value. This is defined as the of epochs needed to reach within 5% of the final objective value.
- **Training time**
- **Stability** - variance and standard deviation in performance metrics across multiple runs.
- **Model complexity** - number of trainable parameters in the model.
- **Jump count** - only measured for stochastic problem 1 (Lyapunov optimisation) - The number of large service-rate changes in stochastic control problems, which measures policy smoothness. A lower jump count indicates more stability and consistency.
- **Time complexity** - theoretical analysis of algorithmic complexity based on model architecture, such as the number of operations in the forward and backward passes. However, it must also be considered that some operations may intrinsically be more expensive despite low time complexity, such as spline evaluations.

Most metrics are tracked programmatically and summarised in both tables and graphs, to be used for side-by-side comparison. 5 training runs are performed to assess stability and consistency.

This methodology ensures a fair and rigorous comparison between MLPs and KANs across a diverse range of optimisation problems, highlighting the conditions under which KANs may provide meaningful advantages.

# Linear Optimisation

## Problem 1 – Lower Dimensional

### Description

The first linear optimisation problem considered is a classical two-variable linear program. It was chosen for its simplicity, visual interpretability, and well-defined feasible region, making it ideal for verifying model correctness and assessing basic learning capabilities of neural network-based approaches.

The objective is to maximise a linear function of two variables,  $x_1$  and  $x_2$ , subject to linear inequality constraints:

$$f(x_1, x_2) = 3x_1 + 5x_2 \quad (6.1)$$

$$\begin{aligned} 2x_1 + 3x_2 &\leq 12 \\ 2x_1 + x_2 &\leq 8 \\ x_1 &\geq 0, \quad x_2 \geq 0 \end{aligned}$$

The formulation models a very simple resource allocation scenario with interpretable geometric constraints and is well-suited for visualisation and validation. To establish a ground truth for comparison, the problem is first solved analytically using `scipy.optimize.linprog` with the HiGHS solver [24], providing the exact solution. The structure of the 2 models used to approximate the optimal objective value are as follows:

- **MLP:**
  - Architecture: 2 input neurons, two hidden layers of 64 neurons each, with ReLU activations, and a single linear output neuron.
  - Loss function: Mean squared error between predicted and true objective values, trained only on feasible points.
  - Training strategy: 5 independent training runs, each for 20,000 epochs, using the Adam optimiser with a learning rate of 0.01.
  - Data: 5000 feasible data samples are generated by randomly sampling the  $[0,6]^2$  domain and retaining only those points satisfying all constraints.
- **KAN:**
  - Architecture: Grid-based KAN using B-spline activations on edges with width [2, 4, 4, 1] and grid resolution 5.

- Training: supervised with the same dataset as the MLP. The loss function is also MSE.
- For each of the 5 runs, only 1000 epochs of training are conducted, and a grid refinement step is in place at 500 epochs. This increases the grid resolution from 5 to 10.
- The grid refinement step allows for the model to leverage local spline interpolation to fit complex functional behaviour more efficiently.

The problem and final predicted results are visualised here:

	<b>X1</b>	<b>X2</b>	<b>Objective</b>
<b>MLP</b>	0.1194	3.9115	19.9211
<b>KAN</b>	0.0053	3.9959	20.0332
<b>Analytical</b>	0.0	4.0	20.0

Table 6-1: Summary of final results reached. The KAN clearly reaches a solution that is closer to the true solution than the MLP.

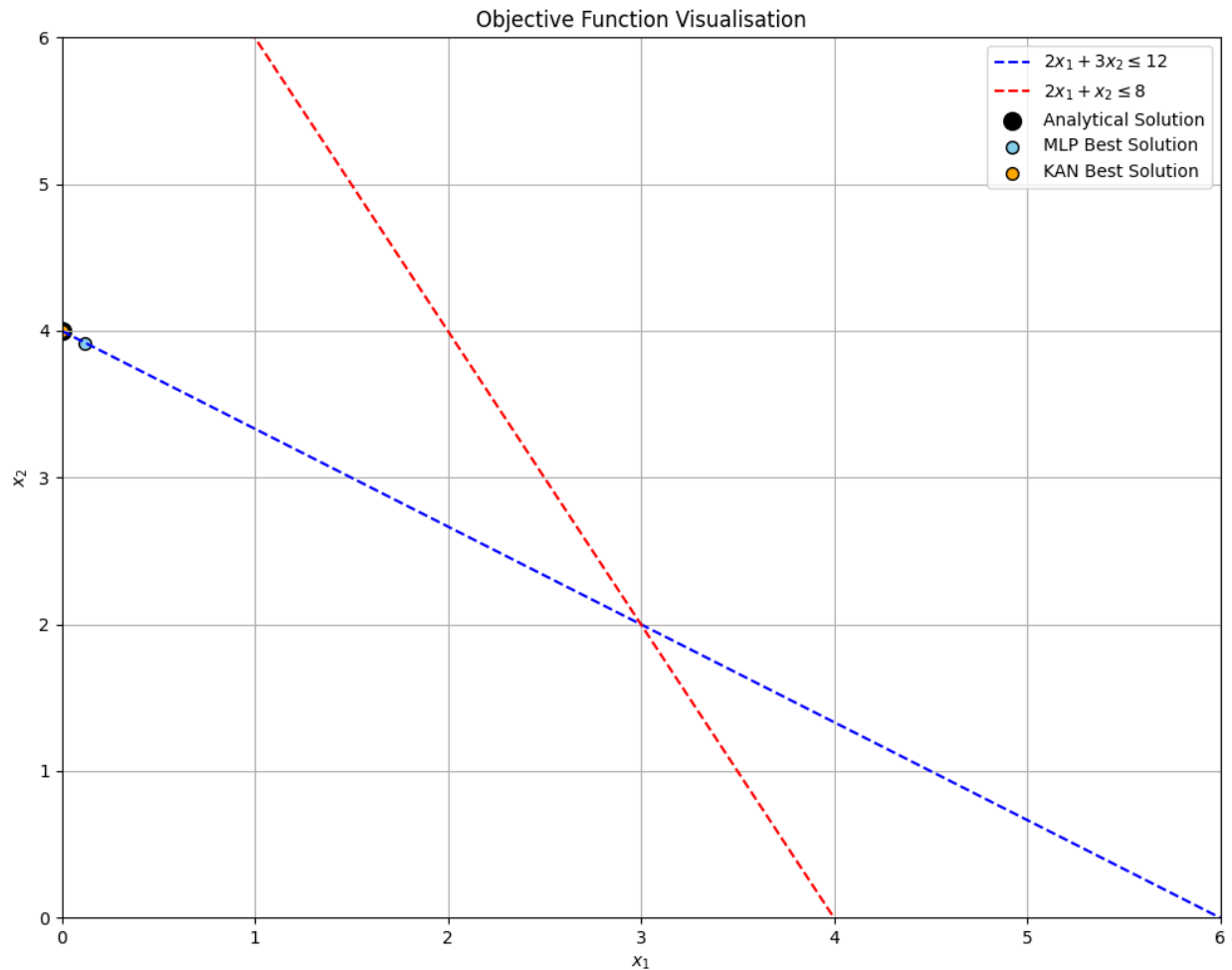


Figure 6-1: Visualisation of the solutions reached by each model. The KAN is shown to reach closer to the actual solution.

## Analysis

The models both performed well, and although the problem being considered in this case is simple, some insights can be made from the performance analysis and metrics comparison.

Metric	MLP	KAN
Mean Objective Value	19.857101	19.902323
Std Objective Value	0.065154	0.120625
Best Objective Value	19.921148	20.059498
Mean Final Loss	0.000045	0.001972
Std Final Loss	0.000059	0.000771
Mean Time (s)	99.284823	120.963200
Std Time (s)	5.821296	1.663264
Mean Convergence Epoch	11148.600000	970.800000
Std Convergence Epoch	7354.543564	6.305553
Model Parameters	4417.000000	604.000000

Table 6-2: Metrics comparison table.

### Objective value performance

Both models could closely approximate the analytical solution, achieving high mean objective values. The KAN slightly outperformed the MLP on average, achieving a higher mean objective value of 19.90 compared to the MLP's 19.86. Moreover, the best solution found by the KAN reached 20.05, being marginally better than the MLP's best of 19.92. This demonstrates the KAN's stronger capacity to capture the optimal solution in this relatively simple problem setup.

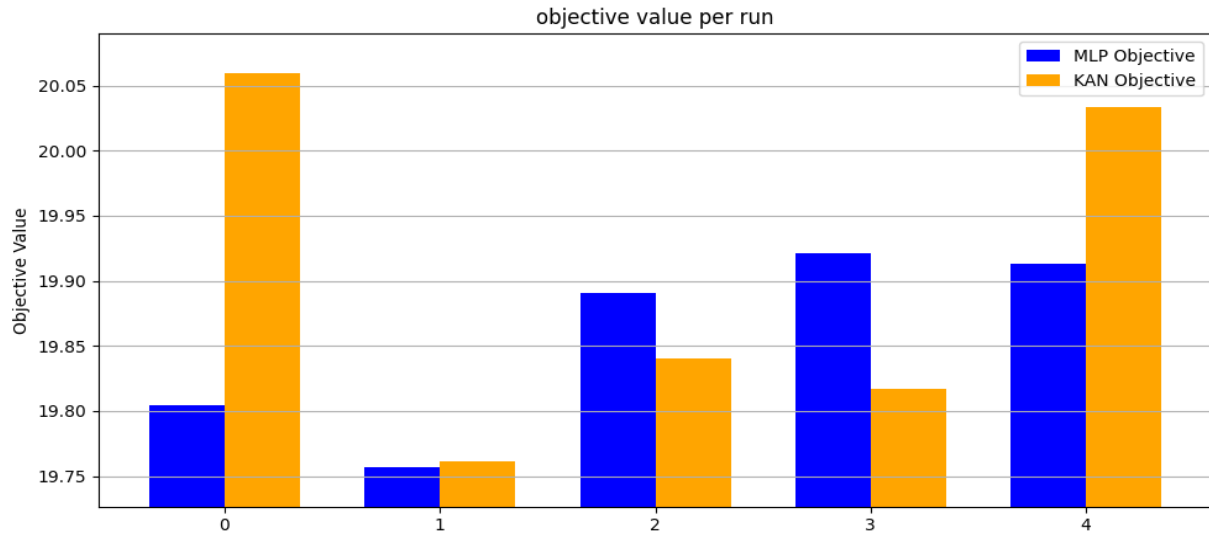


Figure 6-2: Objective values reached in each of the 5 runs for each model.

Figure 6-2 shows the objective values achieved by each model in each of the 5 runs. The MLP model seems to achieve a more consistent objective value, whereas the KAN's objective value varies significantly. This is possibly due to the KAN's structure allowing for more variation in  $x_1$  and  $x_2$  values, leading to more variable objective values.

## Loss and convergence behaviour

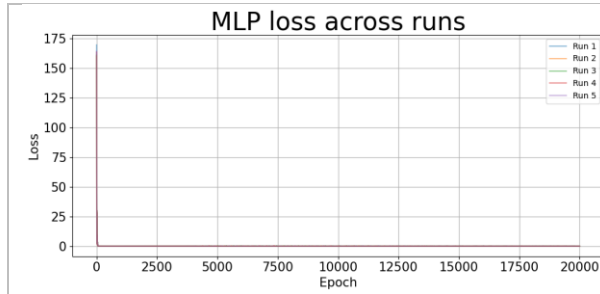


Figure 6-3: MLP loss curve over 20000 epochs.

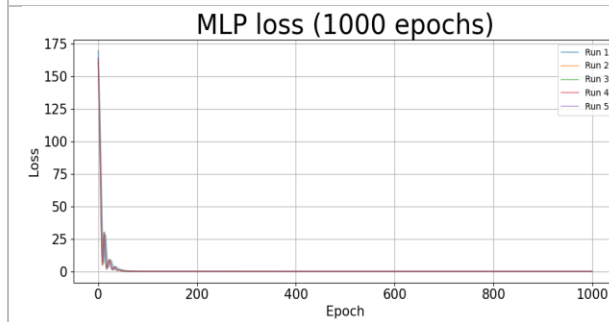


Figure 6-4: MLP loss curve over 1000 epochs.

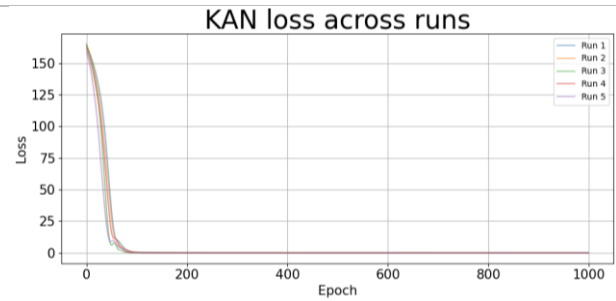


Figure 6-5: KAN loss curve over 1000 epochs.

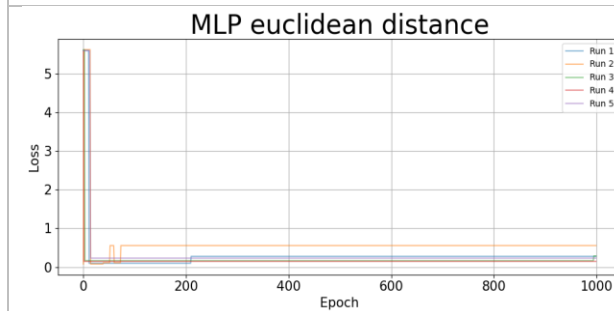


Figure 6-6: MLP L2 norm over 1000 epochs.

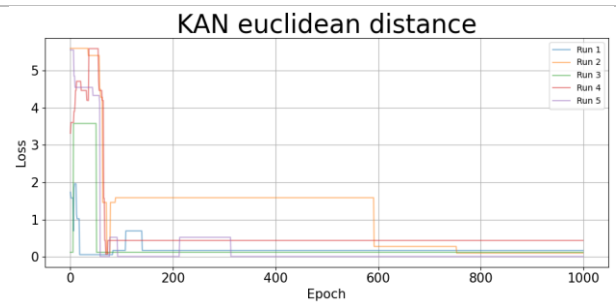


Figure 6-7: KAN L2 norm over 1000 epochs.

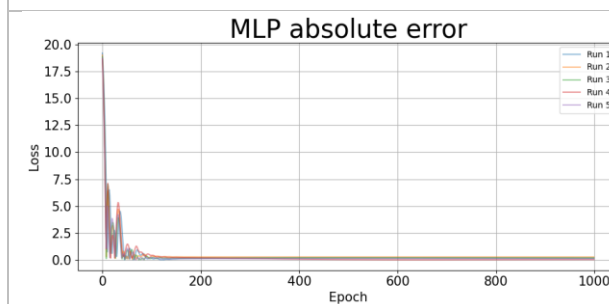


Figure 6-8: MLP absolute error over 1000 epochs.

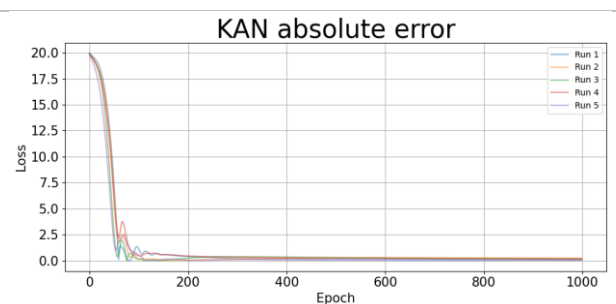


Figure 6-9: KAN absolute error over 1000 epochs.

The loss graphs 6-4 and 6-5 demonstrate how the KAN consistently has a smooth decrease in loss, whereas the MLP has a sharp decrease and oscillates heavily before converging. This may be due to the learning rate being too high, but even so, the MLP reaches a lower loss faster than

the KAN. However, as the below graphs show, the KAN overtakes the MLP in loss at approximately 500 epochs, whereas the MLP plateaus quickly.

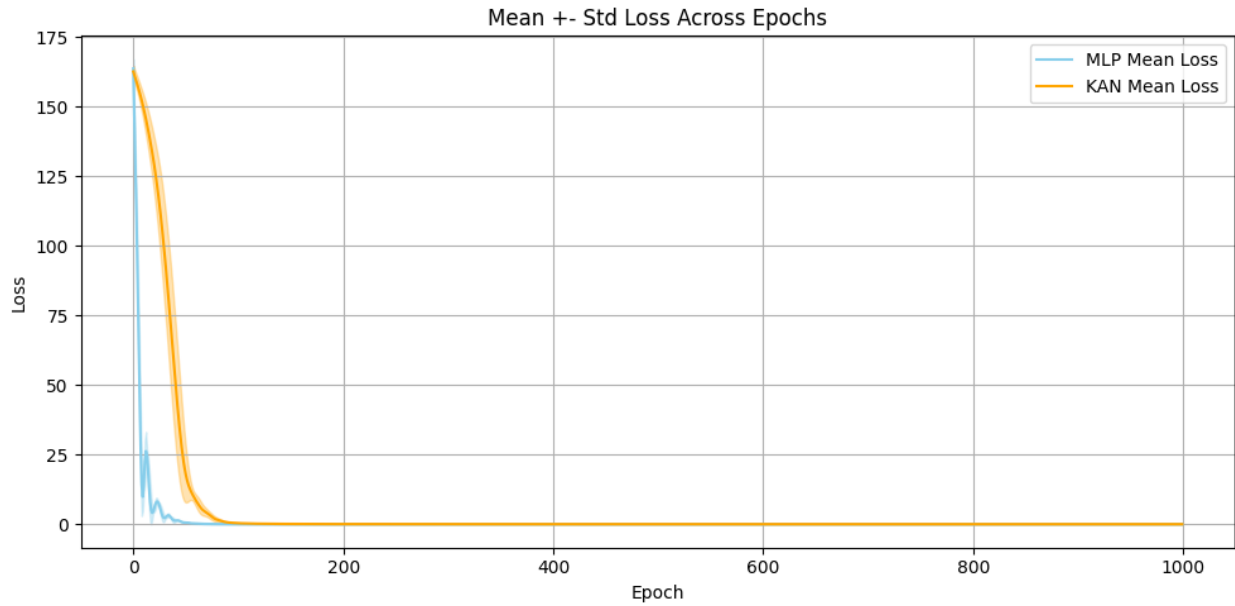


Figure 6-10: Loss curves of both models overlayed.

From the final results, the MLP achieved a much lower mean final loss of  $4.5 \times 10^{-5}$ , compared to the KAN's  $1.97 \times 10^{-3}$ . This suggests that, despite achieving slightly lower objective values, the MLP fit the training data more precisely according to the MSE loss. However, the convergence time differed dramatically between models: the KAN converged in an average of 971 epochs, while the MLP required approximately 11,149 epochs. This indicates that the KAN model is far more efficient at reaching near-optimal solutions in fewer iterations.

## Error observations

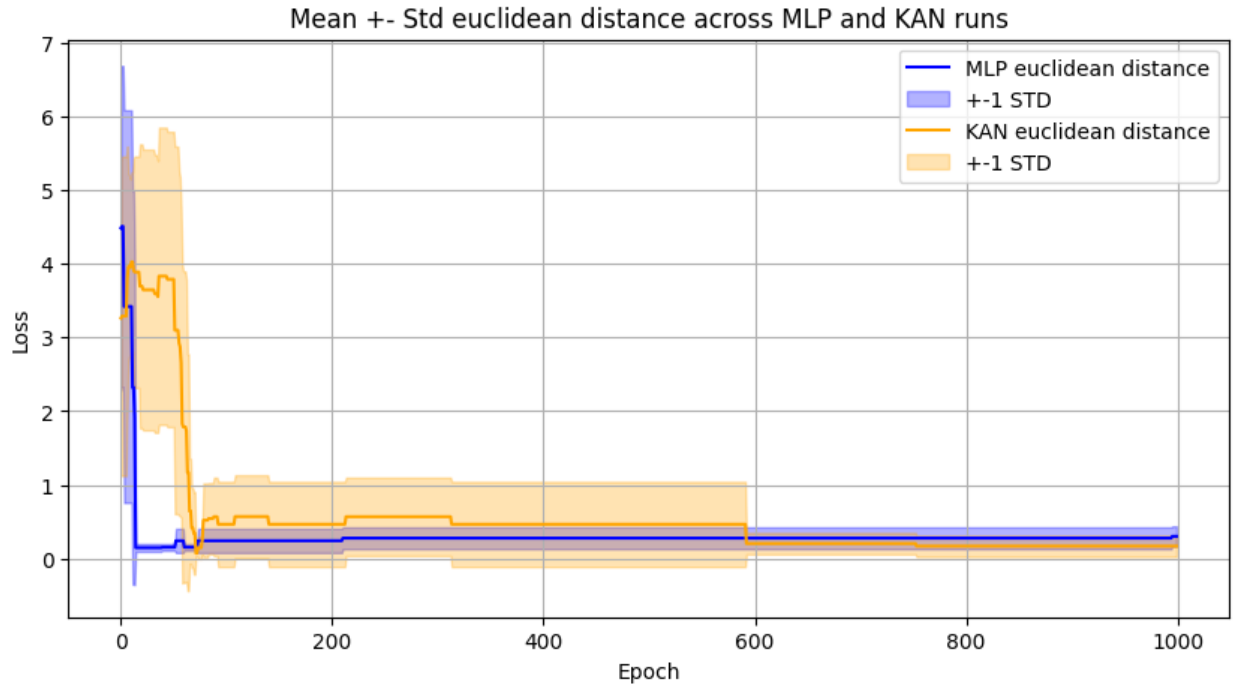


Figure 6-11: L2 norm over 1000 epochs for both models.

What is notable is the fact that the KAN model has a highly variable L2 norm. This could indicate that the KAN is able to explore different areas of the search space more rapidly compared to the MLP, which implies that it could suffer less from falling into local optima. However, since the problem is simpler in scope, this does not provide as much advantage in this case, leading to the MLP obtaining better accuracy measurements within the first 100 epochs.

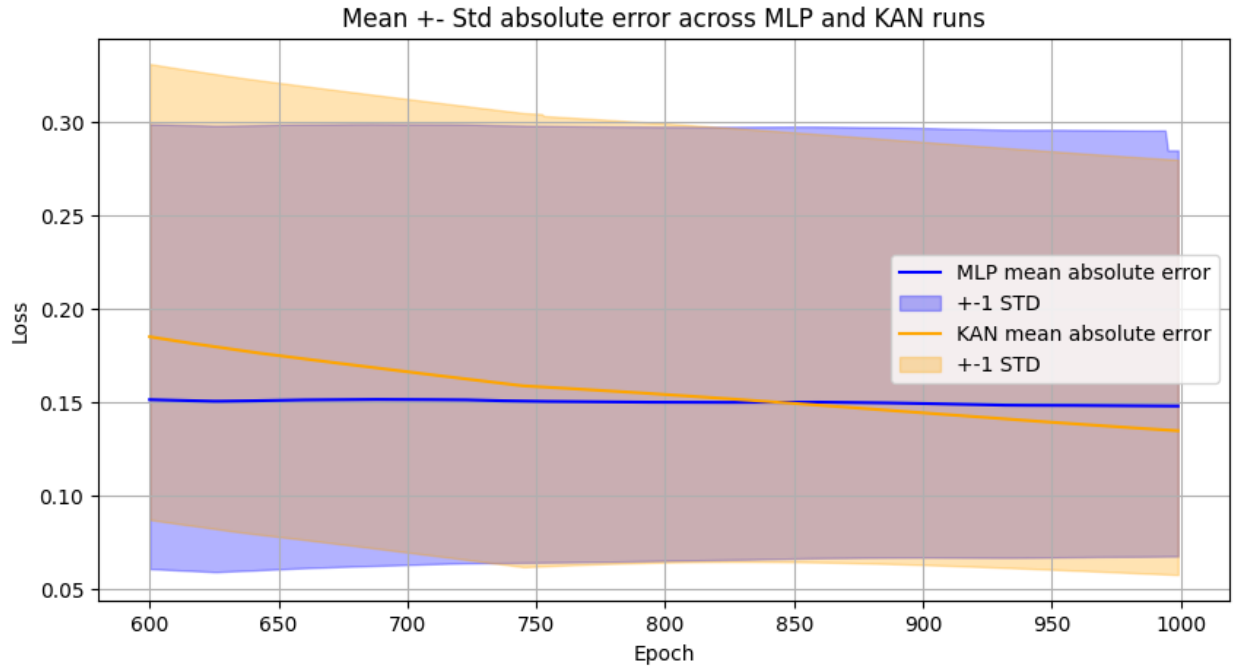


Figure 6-12: Absolute error for both models, restricted to 600-1000 epochs.

In terms of absolute error, the MLP does show better loss towards the start, but the KAN overtakes the MLP's loss at around 800 epochs, and continues to improve significantly, whereas the MLP improves much slower.

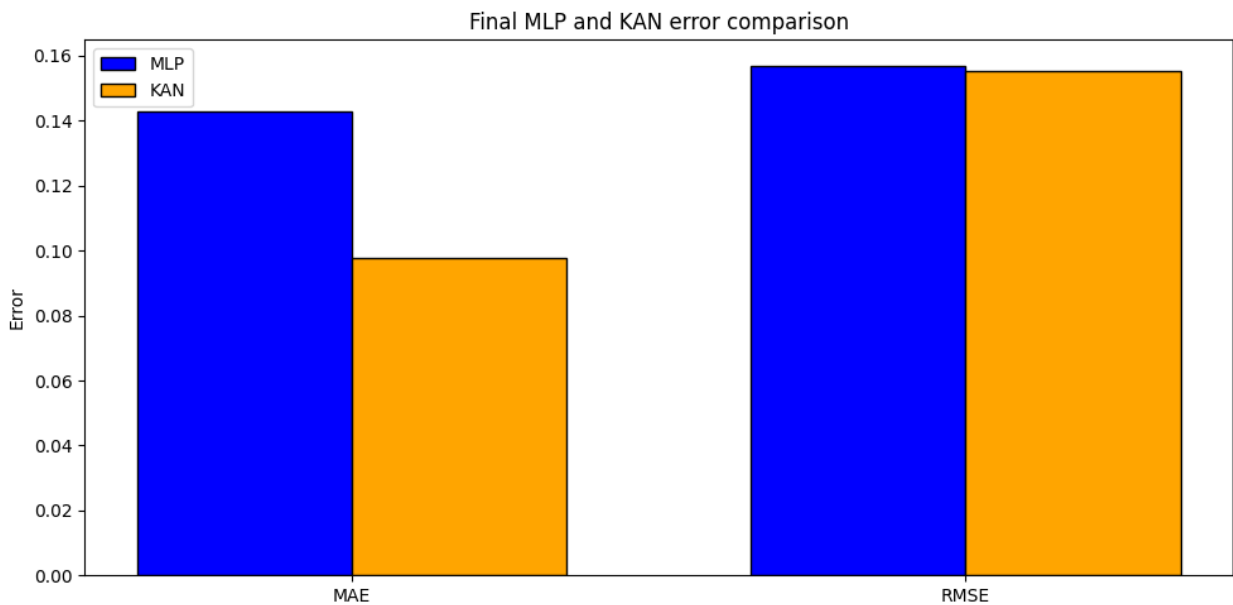


Figure 6-13: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.



In terms of MAE, the KAN clearly outperforms the MLP, although both are nearly equivalent in terms of RMSE, implying that both models are nearly equally as sensitive to larger errors in this case.

### Stability and variability

The KAN exhibited greater variability in objective value, with a standard deviation of 0.12, compared to 0.07 for the MLP, evident in figures 6-3 to 6-9. Similarly, its final loss standard deviation was higher, implying that while the KAN generally converges faster, its performance across runs for this problem is slightly less consistent. In contrast, the MLP was more stable across its multiple training runs but required significantly more training time to do so.

### Training time and model complexity

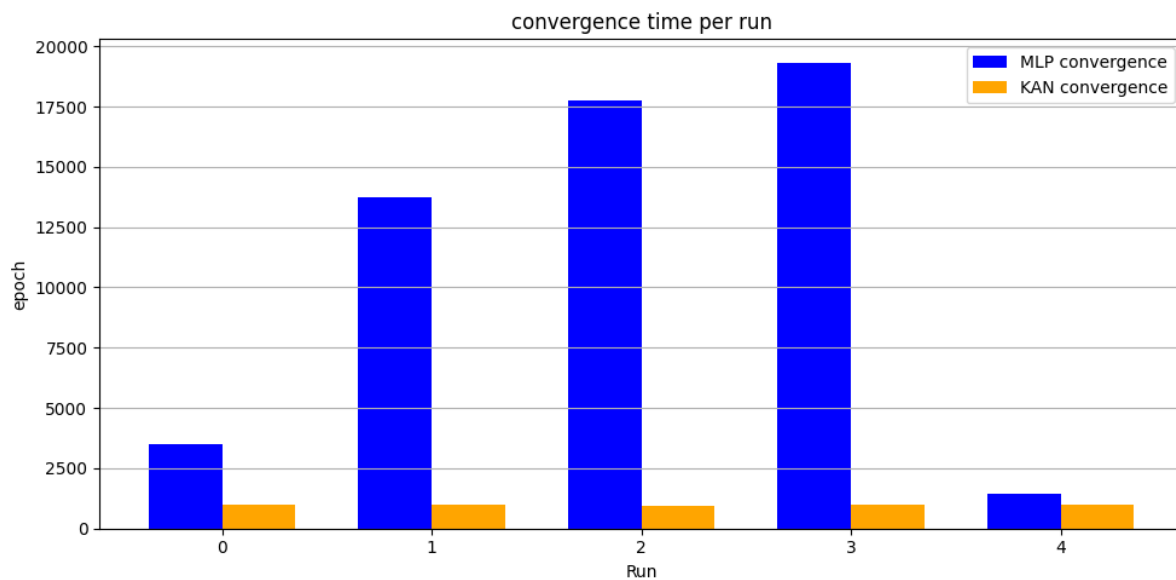


Figure 6-14: Graph of convergence times across all runs for both models.

In terms of time complexity, both models have a theoretical complexity of  $O(N \cdot P)$ , where  $N$  is the number of training samples and  $P$  is the number of model parameters. The KAN's architectural simplicity is reflected in its number of model parameters: 604 for the KAN versus 4,417 for the MLP. Since the number of training samples is constant at 5000 for both, it could be inferred that the KAN is more efficient than the MLP in terms of algorithmic complexity.

However, the KAN has a variable architecture, and contains a grid refinement step at 500 epochs, and considering how the spline computations are more expensive, the real training time is therefore impacted. The MLP required a mean training time of 99.3 seconds per run, whereas the KAN took slightly longer at 121.0 seconds. While the difference is not drastic, it is notable given the KAN's faster convergence epochs, converging to within 5% of the final value within approximately 900 epochs, based on the metrics from table 6-2.

## Search space navigation

Figure 6-15 plots the predictions of coordinates at each epoch by each model, which provides a view into how the solution space is navigated as the models are run. Both models explore the space similarly, by exploring the specific areas which result in the highest objective values, except the KAN is able to explore a wider variety of areas which are not even considered by the MLP in any attempt. This again indicates how effective the KAN model is at exploring the best areas in a search space.

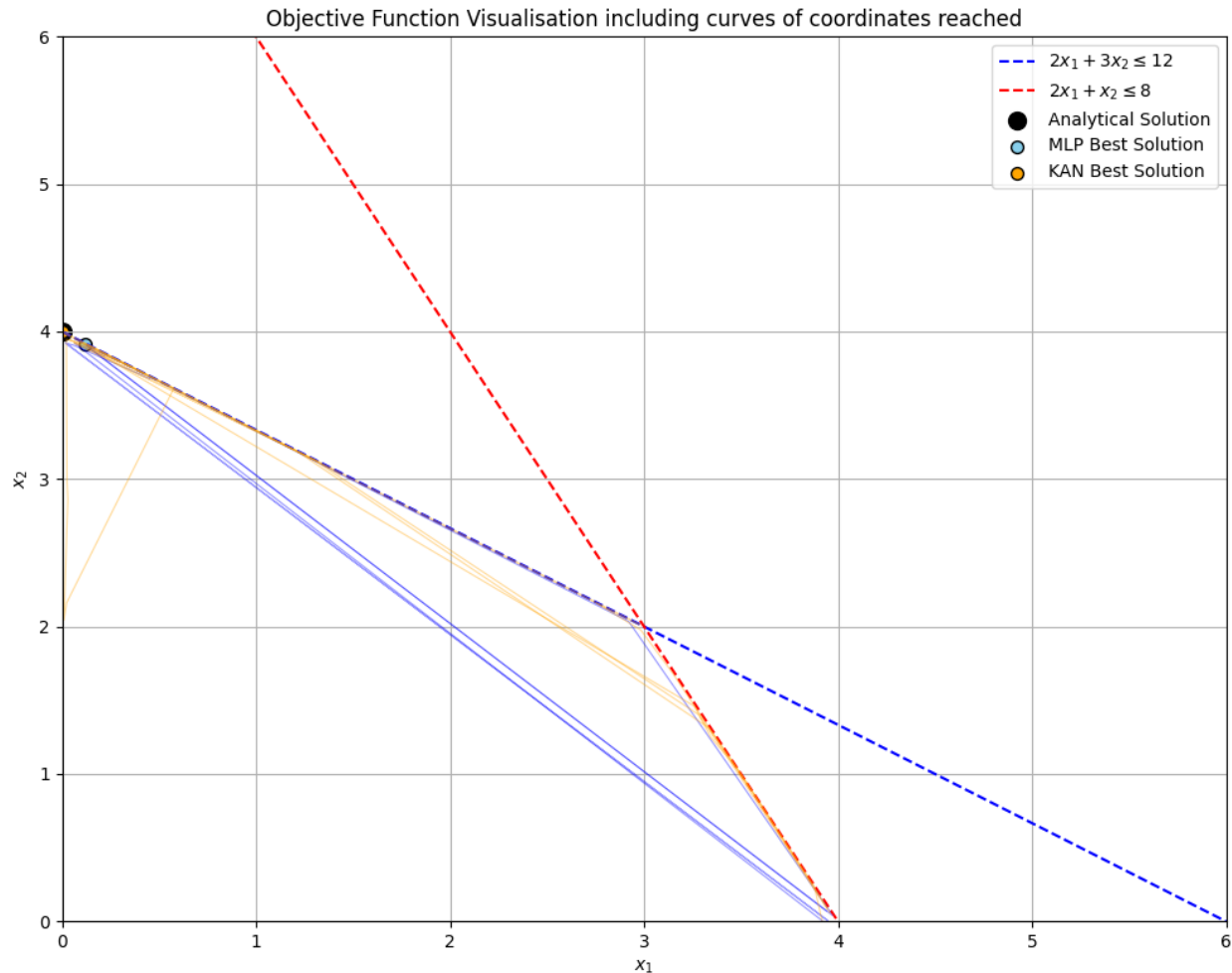


Figure 6-15: Visualisation of the “routes” taken by each model. The orange indicates the KAN’s route, the blue indicates the MLP’s route.

## Summary

Overall, while both models perform well on this problem, the KAN model demonstrates faster convergence, lower model complexity, and marginally better objective values at the cost of slightly higher variability and final loss. The MLP has greater training stability, faster computational time, and a lower final MSE but requires significantly more training epochs to achieve similar levels of objective performance.

## Problem 2 – Higher Dimensional

### Description

The second linear optimisation problem increases the dimensionality and complexity of the task by involving three decision variables and multiple constraints. This problem was designed to test the scalability and generalisation capabilities of the neural networks in higher-dimensional linear optimisation settings.

The objective is to maximise the linear function subject to the system of linear inequalities:

$$f(x_1, x_2, x_3) = 4.2x_1 + 3.7x_2 + 5.5x_3 \quad (6.2)$$

$$1.5x_1 + 2.3x_2 + 3.1x_3 \leq 14.8$$

$$3.2x_1 + 1.8x_2 + 2.5x_3 \leq 12.3$$

$$2.7x_1 + 3.6x_2 + 1.9x_3 \leq 13.5$$

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0$$

This setup allows for more difficult trade-offs to be made between variables due to overlapping constraints in a 3D space. The addition of a third dimension complicates the feasible region, making it more challenging to visualise and solve. The structure of each model is:

- **MLP:**

- Architecture: 3 input neurons, two hidden layers of 64 neurons each (ReLU activations), and a single linear output neuron (3–64–64–1).
- Loss function: MSE between predicted and true objective values. The targets are standardised to improve convergence stability.
- Training strategy: The model is trained using the AdamW optimiser with weight decay for better generalisation. Training is conducted over 20,000 epochs across 5 independent runs.

- **KAN:**

- Architecture: Grid-based KAN using B-spline activations on edges with width [3, 6, 6, 1] and an initial grid resolution of 5, going to 10 at 500 epochs.
- Training strategy: The same dataset and MSE loss function used for the MLP are also used here. Grid refinement is applied similarly to progressively increase resolution and model fidelity.
- Constraint handling: A penalty is applied during training to discourage predictions that fall outside the feasible domain, guiding the model to stay within valid solution bounds.

The best solutions found are visualised below:

	<b>X1</b>	<b>X2</b>	<b>X3</b>	<b>Objective</b>
<b>MLP</b>	0.2194	0.1020	4.5241	26.1795
<b>KAN</b>	0.1677	0.4057	4.3663	26.2161
<b>Analytical</b>	0.18314425	0	4.68557536	26.5399

Table 6-3: Summary of final results reached. The KAN again manages to reach a better solution.

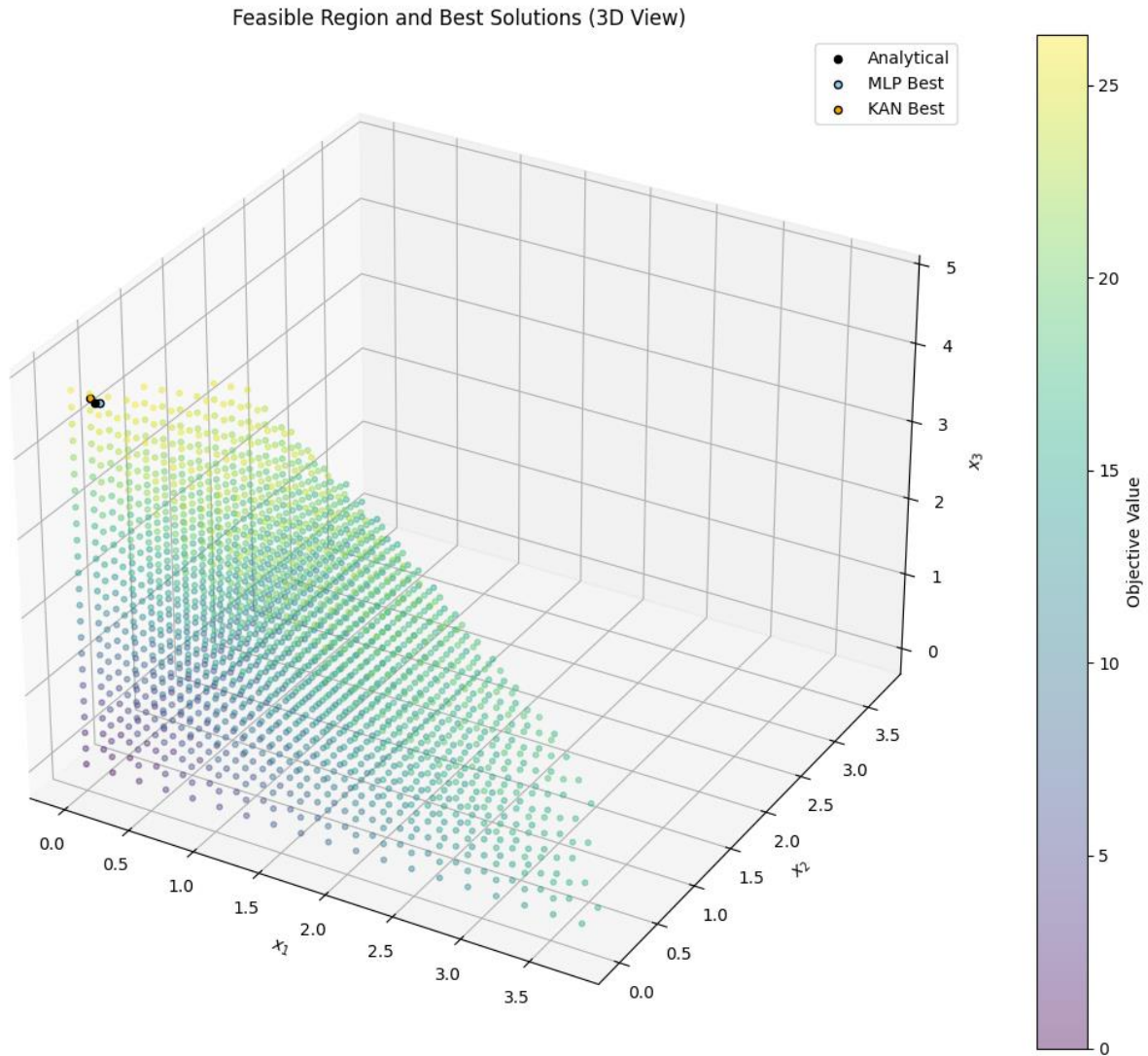


Figure 6-16: Graph of the feasible region, coloured to show where the best objective values are. Both models perform well here.

## Analysis

The results of Linear Problem 2 demonstrate notable differences between the MLP and KAN models in terms of both optimisation quality and convergence behaviour.

Metric	MLP	KAN
Mean Objective Value	25.897978	25.969235
Std Objective Value	0.291954	0.191505
Best Objective Value	26.220579	26.216091
Mean Final Loss	0.012044	0.001736
Std Final Loss	0.024073	0.001457
Mean Time (s)	71.075736	114.207042
Std Time (s)	11.525182	6.458222
Mean Convergence Epoch	14018.200000	975.000000
Std Convergence Epoch	7338.979041	3.405877
Model Parameters	4481.000000	1567.000000

Table 6-4: Metrics comparison table

### Objective value performance

The KAN achieved a slightly higher mean objective value of 25.969 compared to the MLP with 25.898, suggesting a marginally better approximation of the optimal solution. The best solutions achieved by both models were very close, indicating that both methods were capable of reaching near-optimal points under favourable conditions. However, the KAN's lower standard deviation of 0.192 compared to 0.292 highlights that it achieved consistently strong solutions across different runs, whereas the MLP exhibited greater variability in this case.

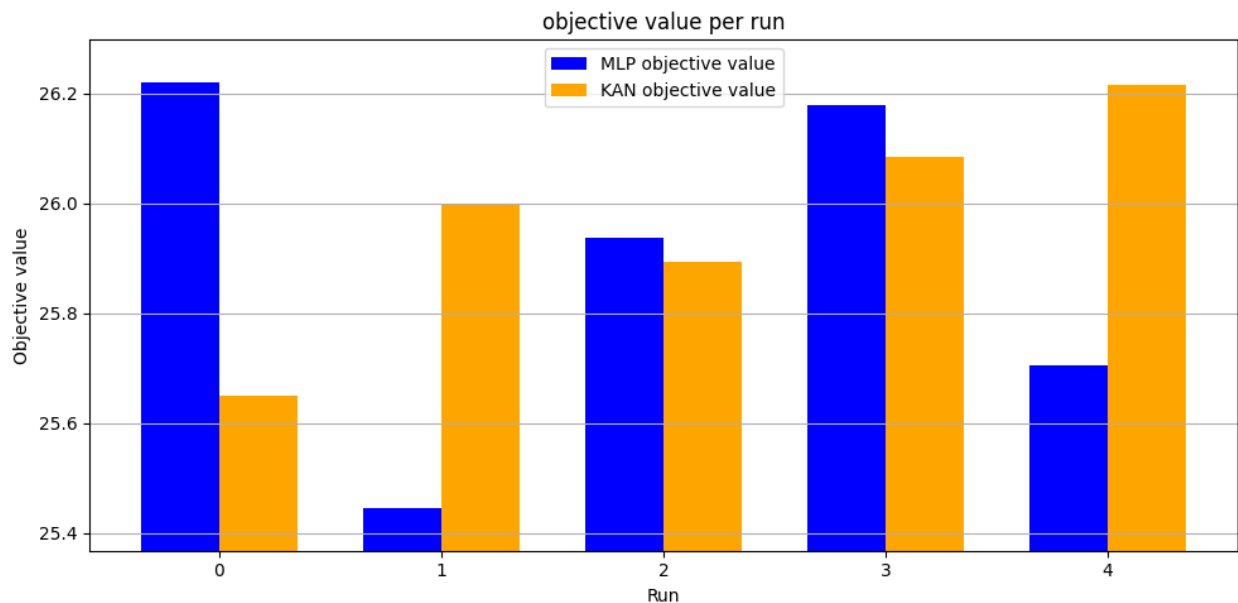


Figure 6-17: Objective values reached in each of the 5 runs for each model.

Across runs, the KAN surprisingly achieved more consistent results than the MLP, supported by the lower standard deviation in loss as well.

### Loss and convergence behaviour

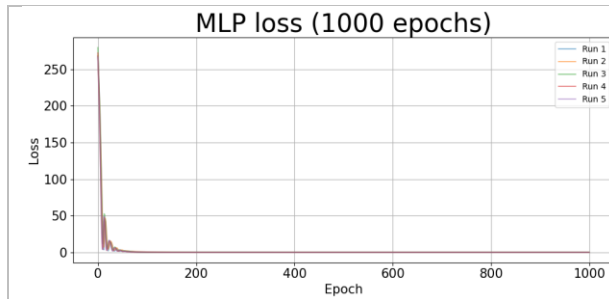


Figure 6-18: MLP loss curve over 1000 epochs.

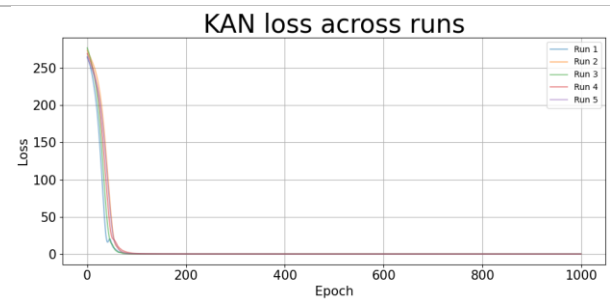


Figure 6-19: KAN loss curve over 1000 epochs.

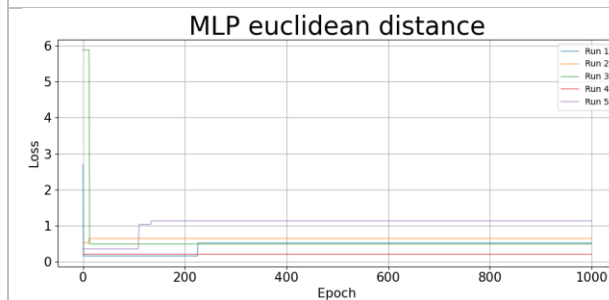


Figure 6-20: MLP L2 norm over 1000 epochs.

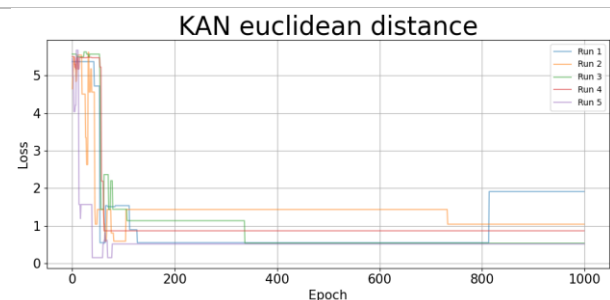


Figure 6-21: KAN L2 norm over 1000 epochs.

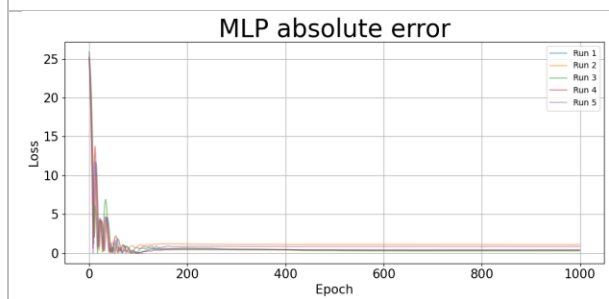


Figure 6-22: MLP absolute error over 1000 epochs.

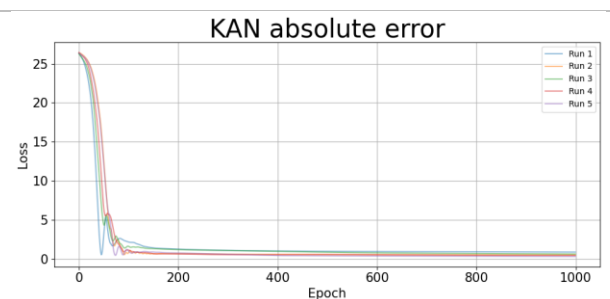


Figure 6-23: KAN absolute error over 1000 epochs.

The loss graphs above again demonstrate the KAN's smooth decrease in loss, and the MLP's oscillation due to the relatively high learning rate, similar to problem 1.

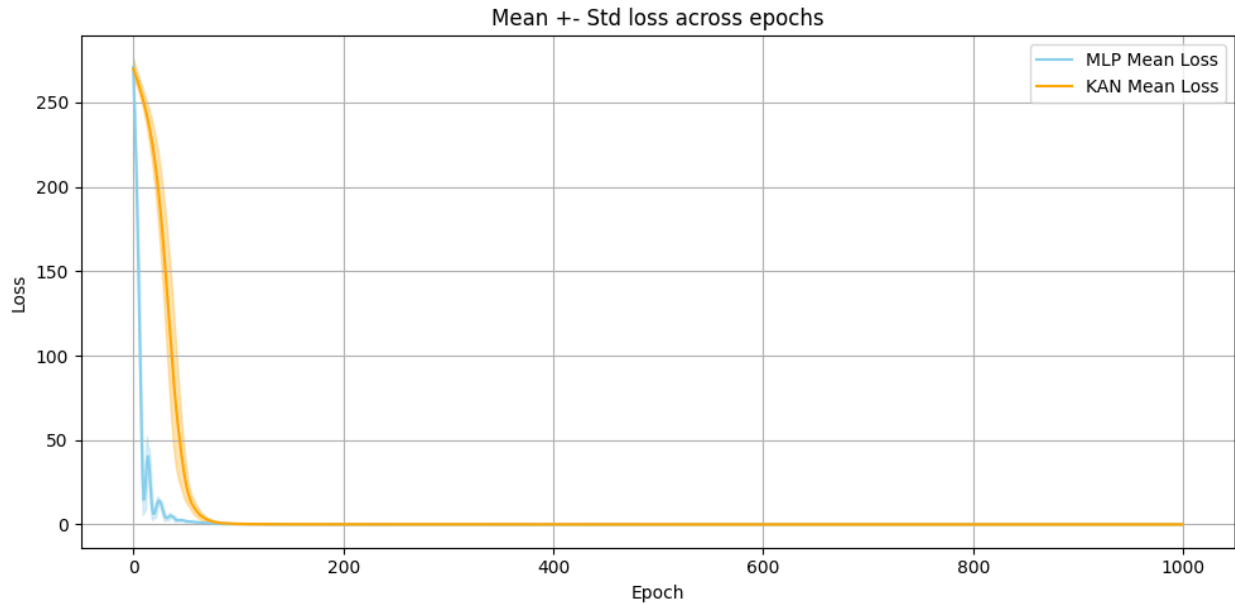


Figure 6-24: Loss curves of both models overlayed.

The KAN attained a significantly lower mean final training loss of 0.0017 compared to the MLP at 0.0120. Furthermore, the standard deviation of final loss for the KAN was much smaller, indicating that KAN training was not only more accurate but also more stable across multiple runs.

## Error observations

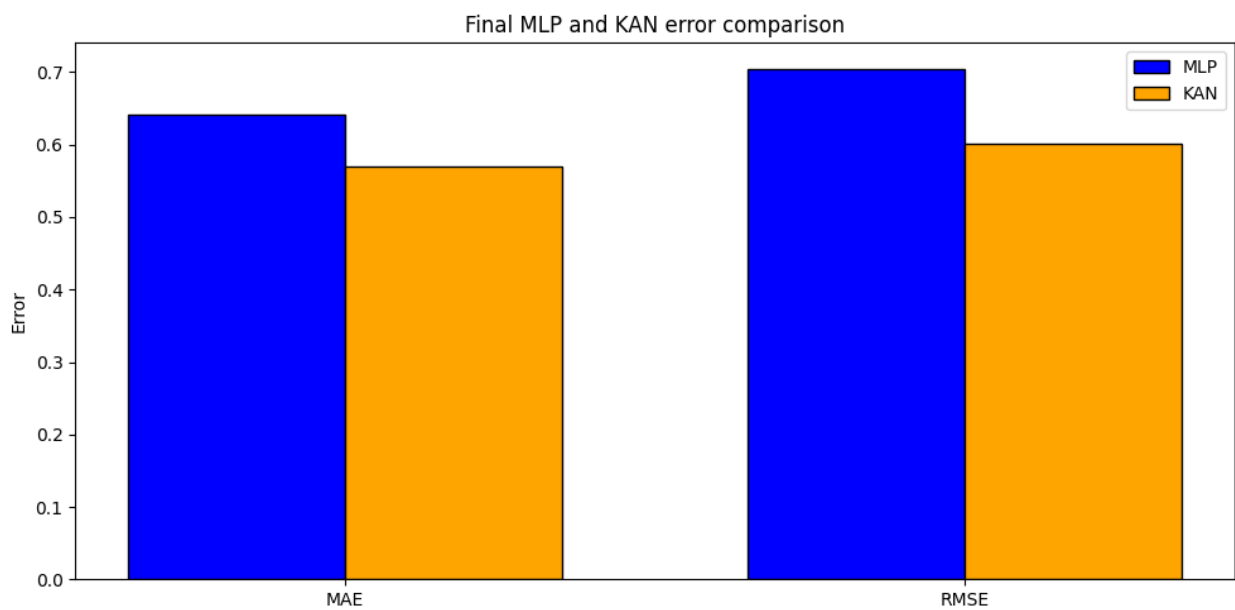


Figure 6-25: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.

It is evident from figure 6-25 that the KAN performed better in terms of final loss, reaching a lower MAE and RMSE error.

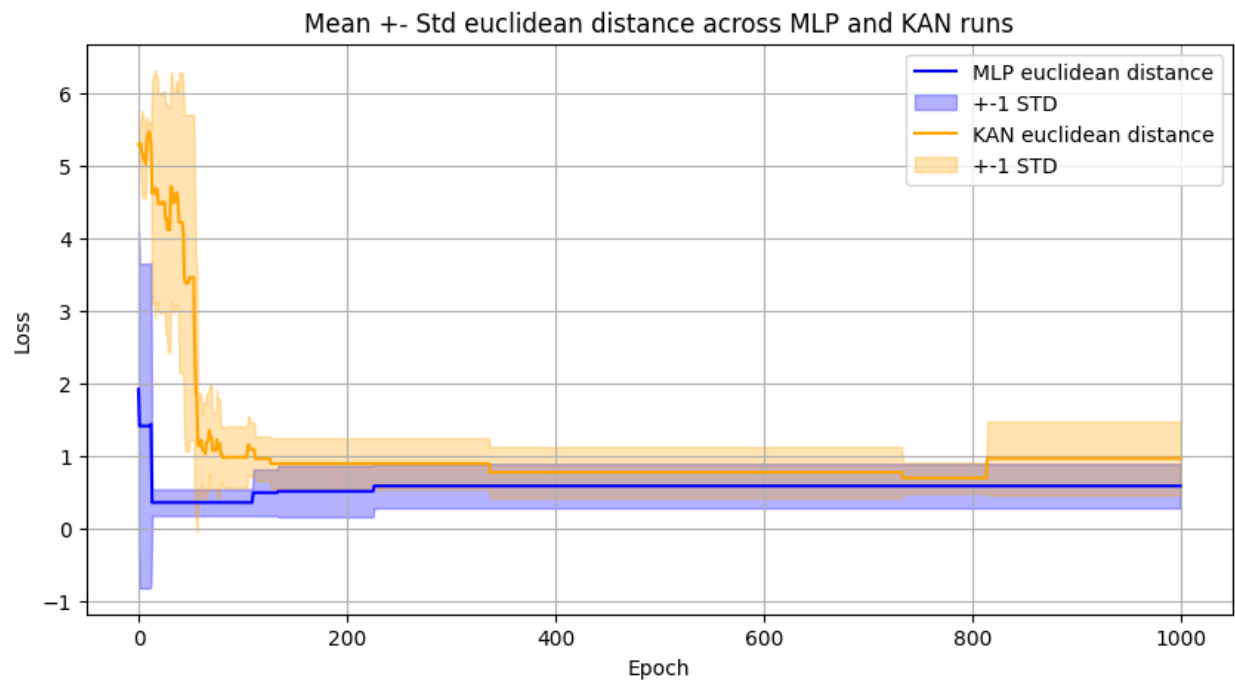


Figure 6-26: L2 norm over 1000 epochs for both models.

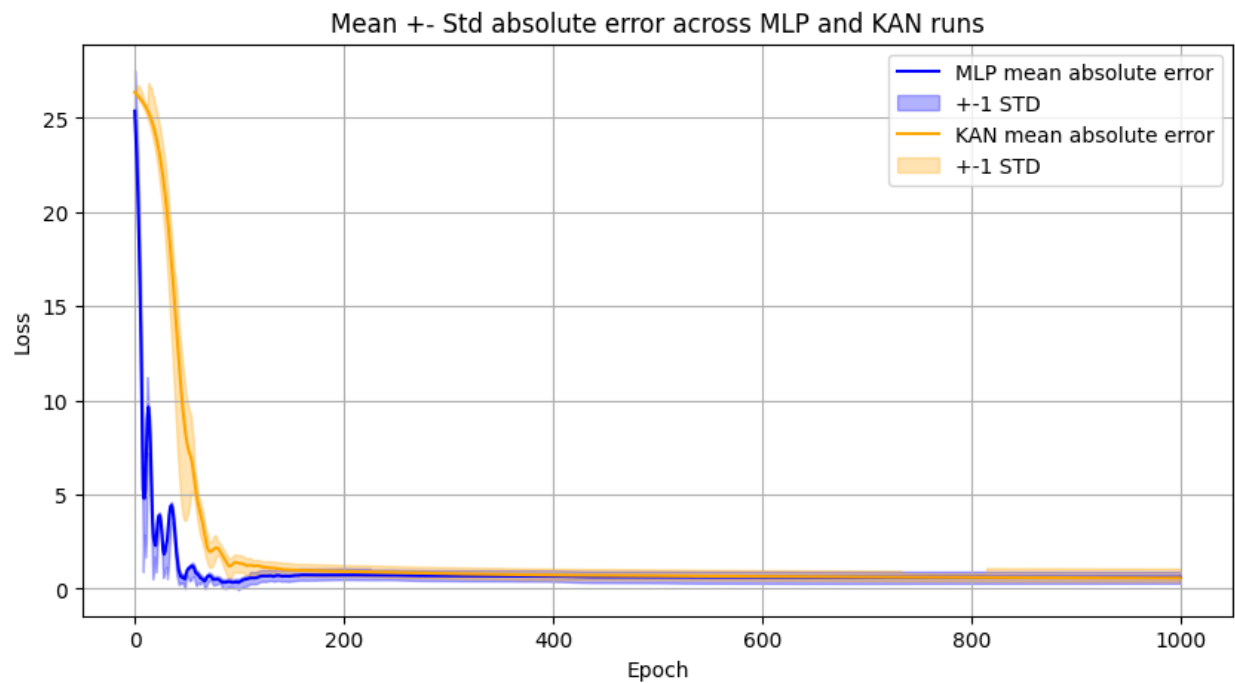


Figure 6-27: Absolute error over 1000 epochs for both models.



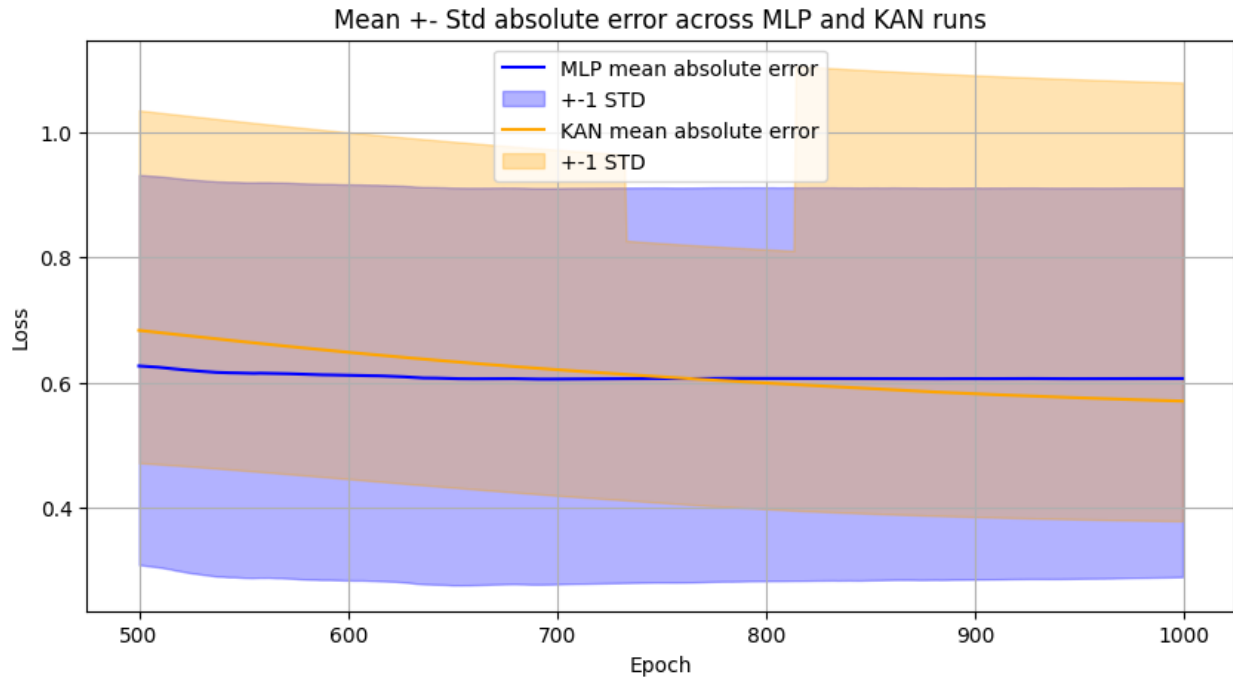


Figure 6-28: Absolute error for both models, restricted to 500-1000 epochs. The KAN overtakes the MLP's curve here.

Based on the L2 norm in figure 6-26, the KAN appears to achieve a worse result. However, in terms of final objective value and absolute error, it performs better, surpassing the MLP at around 750 epochs and continuing to decrease in absolute error faster, showing that it is able to find optimal solutions in the search space more thoroughly.

### Training time and model complexity

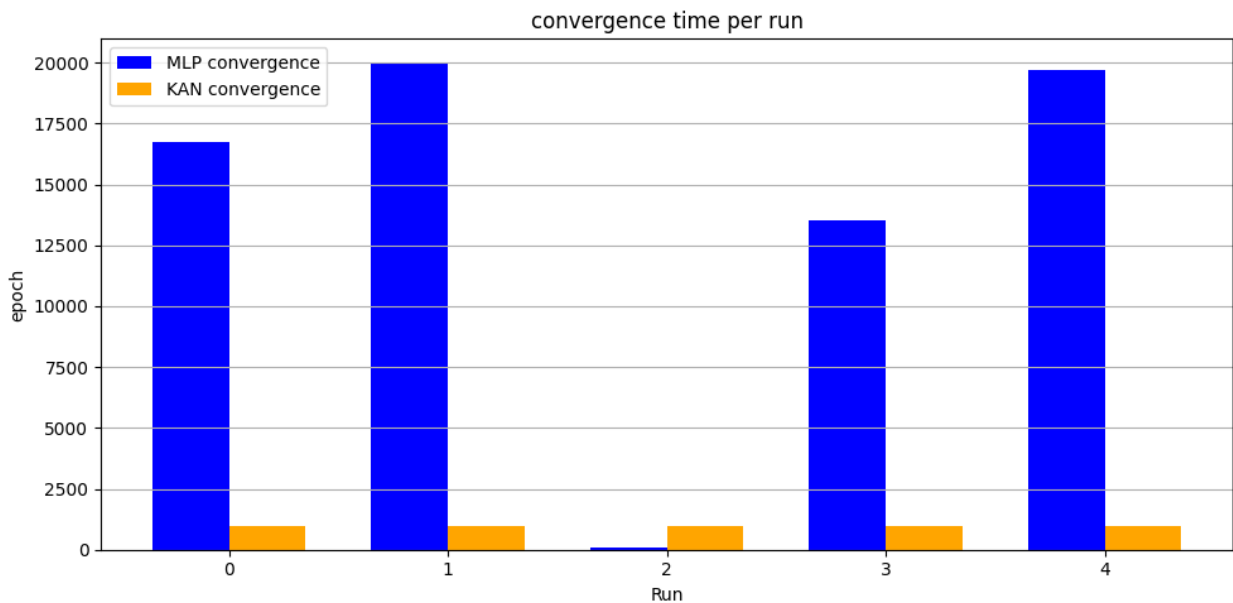


Figure 6-29: Graph of convergence times across all runs for both models.

KAN models converged significantly faster, reaching the convergence threshold in approximately 975 epochs on average, compared to the MLP which took around 14,018 epochs. Additionally, KAN exhibited extremely low variance in convergence epochs, with a standard deviation of 3.4, whereas the MLP showed a very wide spread at 7339, due to the outlier at run 2. The MLP likely managed to converge faster than the KAN during run 2 because of lucky dataset generation.

Despite its generally better performance, the KAN required more computational time on average (114.2 seconds) than the MLP (71.1 seconds). This increase is expected given the complexity introduced by KAN's spline activations and grid management. However, the time difference remained moderate, and both models remained practically efficient for this scale of problem. The time complexity of both models remained the same as in problem 1.

The KAN used significantly fewer parameters than the MLP, using 1567 compared to 4481 for the MLP. While the KAN had a higher number of parameters compared to simpler problems, the performance improvements in stability and convergence justify the additional model complexity.

### **Summary**

In Linear Problem 2, the KAN again demonstrated superior stability, faster convergence, and slightly better objective value approximation compared to the MLP, albeit at the cost of moderately increased training time.

## **Cross-problem Comparison**

When considering how the models perform across the lower dimensional problem 1 and the higher dimensional problem 2, it is evident that the KAN handles the increase in dimensionality better on problem 2, especially when compared to the MLP. On problem 1, both were fairly similar, with each having advantages and disadvantages in different areas. On problem 2, the KAN clearly reaches a more optimal solution in a similar convergence time, in approximately 900 epochs, whereas the MLP's convergence jumps from 11000 to 14000 epochs, more than 10-15 times as much. The KAN also appears to be more stable in problem 2, consistently reaching nearer to the optimal solution than the MLP, even though the MLP uses 3 times as many model parameters. Interestingly, in both problems the MLP initially reaches a lower loss faster, despite using the same learning rate, although the KAN does overtake it eventually.

Overall, the results suggest that the KAN model is better suited for complex linear optimisation tasks with higher dimensionality.

# Integer Optimisation

## Problem 1 – Lower Dimensional

### Description

This problem introduces integer constraints into a standard linear programming setup. The goal is to explore how KANs can handle discrete solution spaces compared to typical neural networks, and to observe how performance differs from relaxed continuous problems.

The objective is to maximise the function subject to the constraints:

$$f(x_1, x_2) = 3x_1 + 2x_2 \tag{7.1}$$

$$\begin{aligned} 4x_1 + 3x_2 &\leq 17 \\ 2x_1 + 5x_2 &\leq 14 \\ x_1, x_2 &\in \mathbb{Z}_{\geq 0} \end{aligned}$$

This formulation models real world optimisation scenarios, such as scheduling or production planning, where solutions must be whole numbers. Due to this requirement, the problem becomes an integer linear program (ILP), which is generally more difficult to solve than its continuous counterpart.

To establish ground truth, the problem is solved both as a relaxed linear program (ignoring the integer constraint) and as a true ILP using `scipy.optimize.milp` [24]. The structures of each model are as follows:

- **MLP:**
  - Architecture: 2 input neurons, two hidden layers of 32 neurons each (ReLU activations), and a single linear output neuron (2–32–32–1).
  - Loss function: MSE between predicted and true objective values, with an added regularisation term that encourages reducing the loss when the mean increases, incentivising the model to predict higher objective values.
  - Training strategy: The model is trained using the AdamW optimiser. To adapt the model for integer optimisation, feasible samples are restricted to integer values.
- **KAN:**
  - Architecture: Grid-based KAN using B-spline activations on edges with width [2, 4, 4, 1].
  - Constraint handling: Training is limited to integer-valued inputs, with the same regularisation penalty added to the loss function.

- Training strategy: To improve the model's ability to represent discrete behaviours, grid refinement is applied at fixed intervals, increasing the grid resolution from 3 to 10 at 500 epochs. This allows the KAN to progressively learn more precise representations over time.

The best solutions found are visualised below:

	<b>X1</b>	<b>X2</b>	<b>Objective</b>
<b>MLP</b>	4	0	20
<b>KAN</b>	4	0	20
<b>Analytical</b>	4	0	20

Table 7-1: Summary of final results reached.

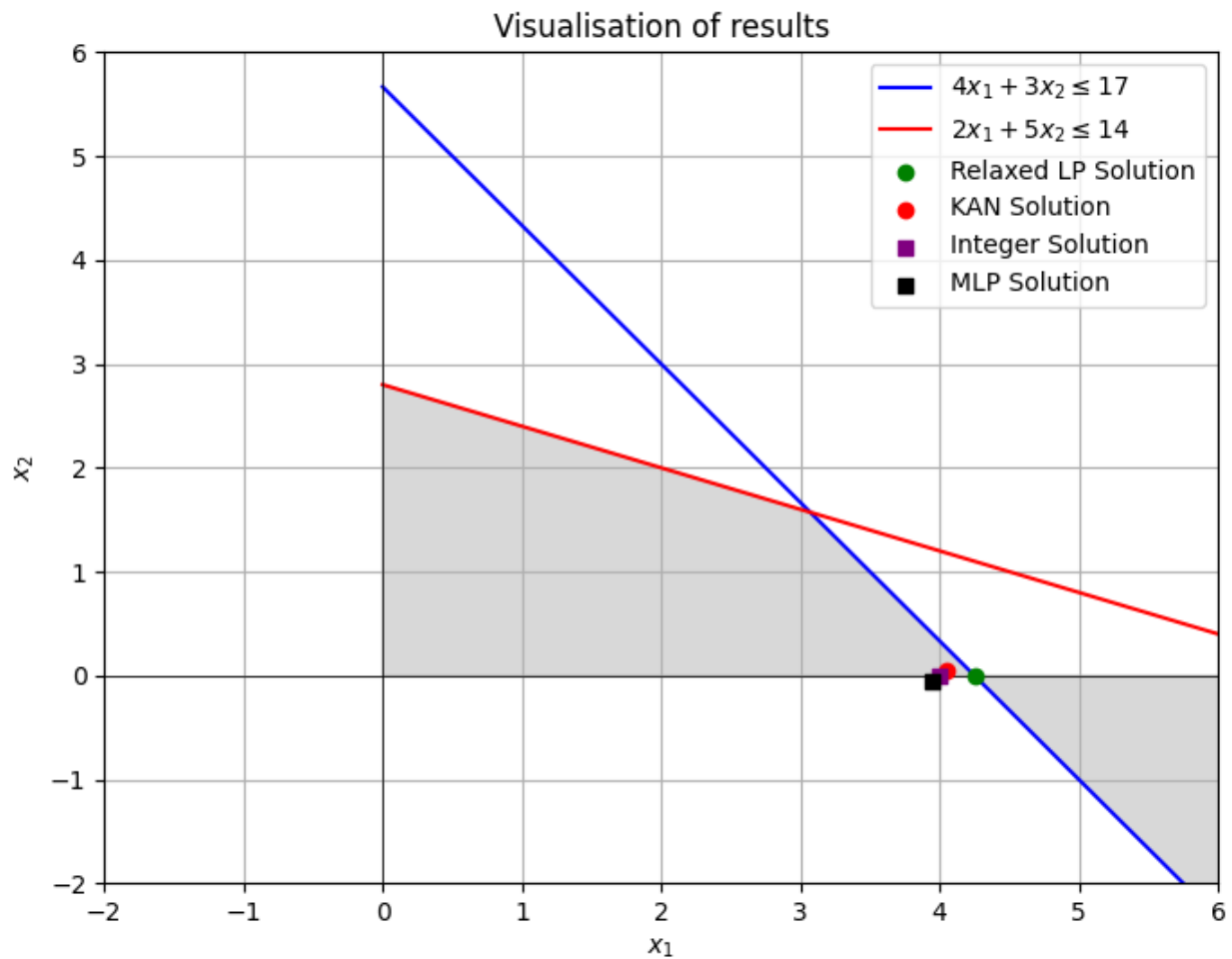


Figure 7-1: Visualisation of the solutions reached by each model. Both models reach exactly the same solution.

## Analysis

Problem 2 tested each model's ability to learn discrete feasible solutions under strict integer constraints. Both the MLP and KAN were able to discover the optimal solution, but their learning dynamics and overall reliability varied in interesting ways.

<b>Metric</b>	<b>MLP</b>	<b>KAN</b>
<b>Mean Objective Value</b>	8.800000	9.200000
<b>Std Objective Value</b>	3.059412	3.310589
<b>Best Objective Value</b>	12.000000	12.000000
<b>Mean Final Loss</b>	0.4975001	7.612226
<b>Std Final Loss</b>	0.0002460	7.150828
<b>Mean Time (s)</b>	76.37156	45.601174
<b>Std Time (s)</b>	13.37965	5.680167
<b>Mean Convergence Epoch</b>	2.000000	531.600000
<b>Std Convergence Epoch</b>	2.449490	265.208295
<b>Model Parameters</b>	1185.000	360.000000

Table 7-2: Metrics comparison table.

### Objective value performance

Both models were able to reach the global optimum of 12 in most runs, but on average, the KAN model performed slightly better overall. The KAN achieved a mean objective value of 9.20, compared to the MLP's mean of 8.80. However, the standard deviation of objective values was higher for the KAN at 3.31 vs 3.06, showing that while it occasionally found better solutions, its results were more spread out compared to the MLP. In contrast, the MLP's performance was slightly more consistent across runs, even if the mean objective was marginally lower.

### Loss and convergence behaviour

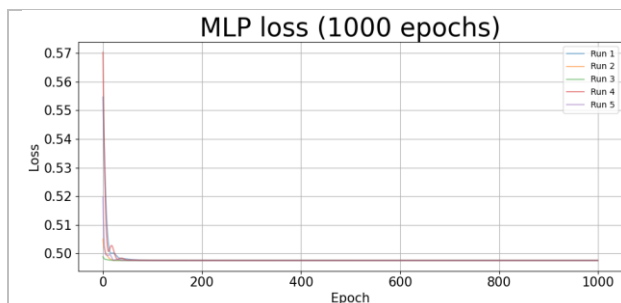


Figure 7-2: MLP loss curve over 1000 epochs.

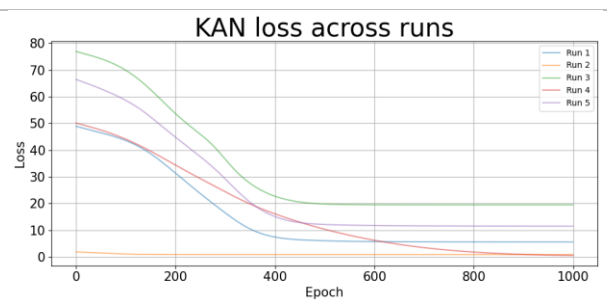


Figure 7-3: KAN loss curve over 1000 epochs.

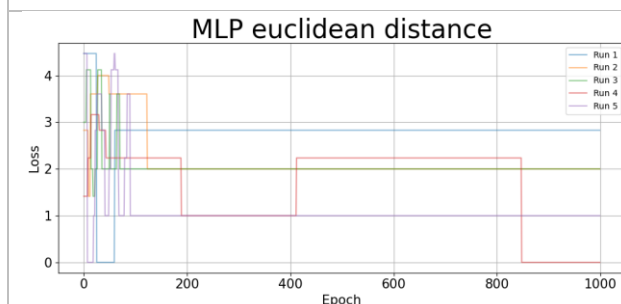


Figure 7-4: MLP L2 norm over 1000 epochs.

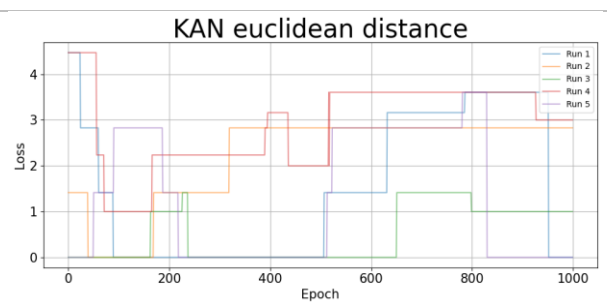


Figure 7-5: KAN L2 norm over 1000 epochs.

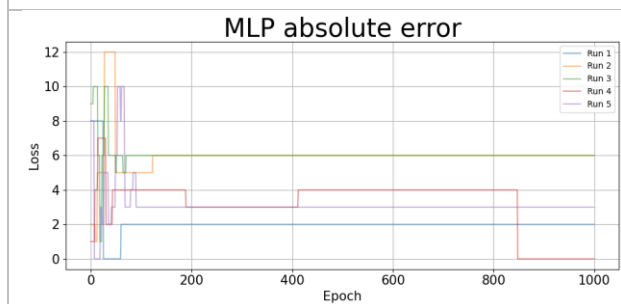


Figure 7-6: MLP absolute error over 1000 epochs.

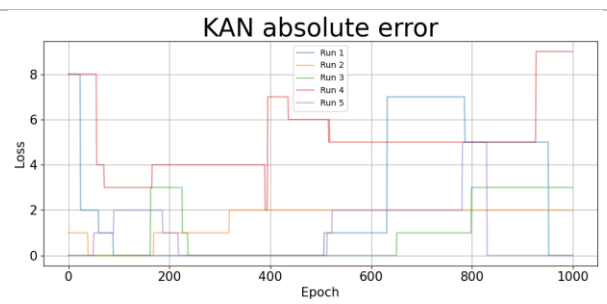


Figure 7-7: KAN absolute error over 1000 epochs.

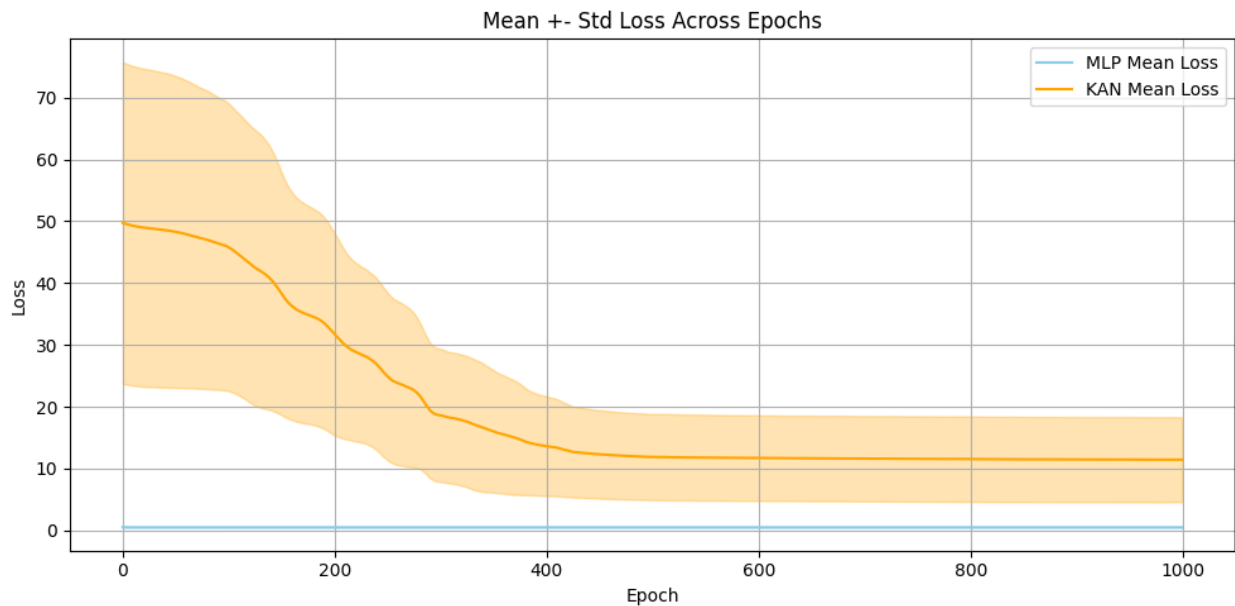


Figure 7-8: Loss curves of both models overlayed.

The MLP achieved a significantly lower mean final training loss (0.4975) compared to the KAN (7.6122). However, this lower loss did not directly translate to better objective outcomes, since the KAN was also able to reach equally adequate solutions. The KAN likely suffered the much higher loss because it was more effective at navigating the integer solution space, which is evident from figures 7-2 to 7-7. Figure 7-8 also shows the KAN’s higher variability.

## Error observations

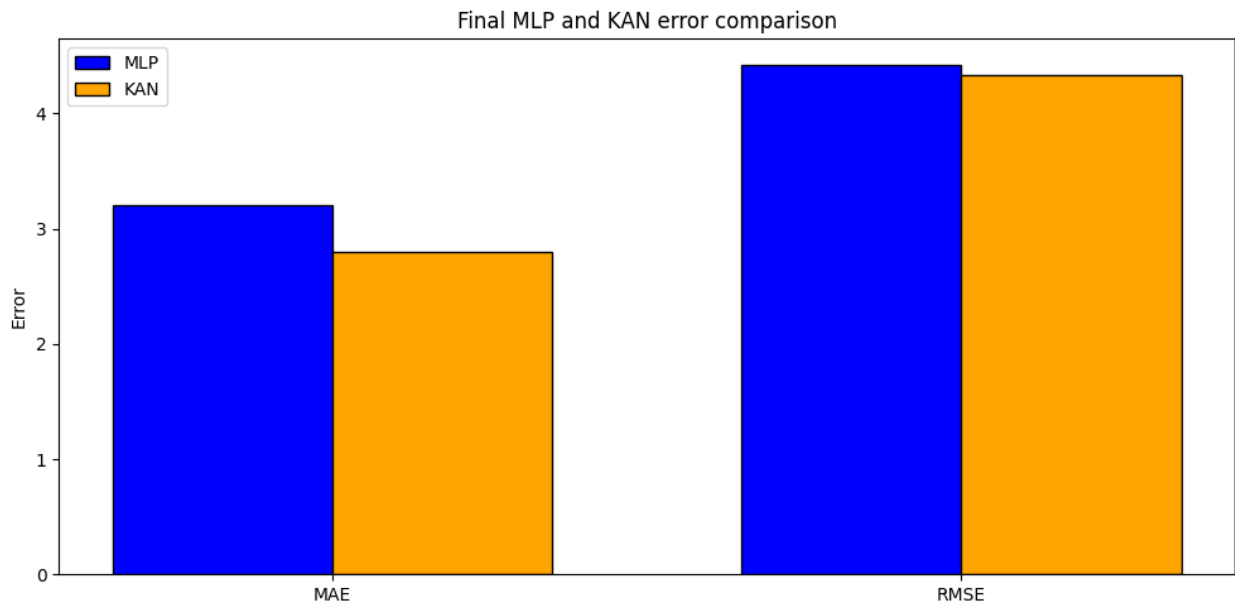


Figure 7-9: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.

The KAN was shown to outperform the MLP significantly in both final MAE and RMSE, from figure 7-9. It also managed to initially reach better L2 norm and absolute error values, although the MLP was able to overtake it in these areas towards the latter half, shown in figures 7-10 and 7-11.

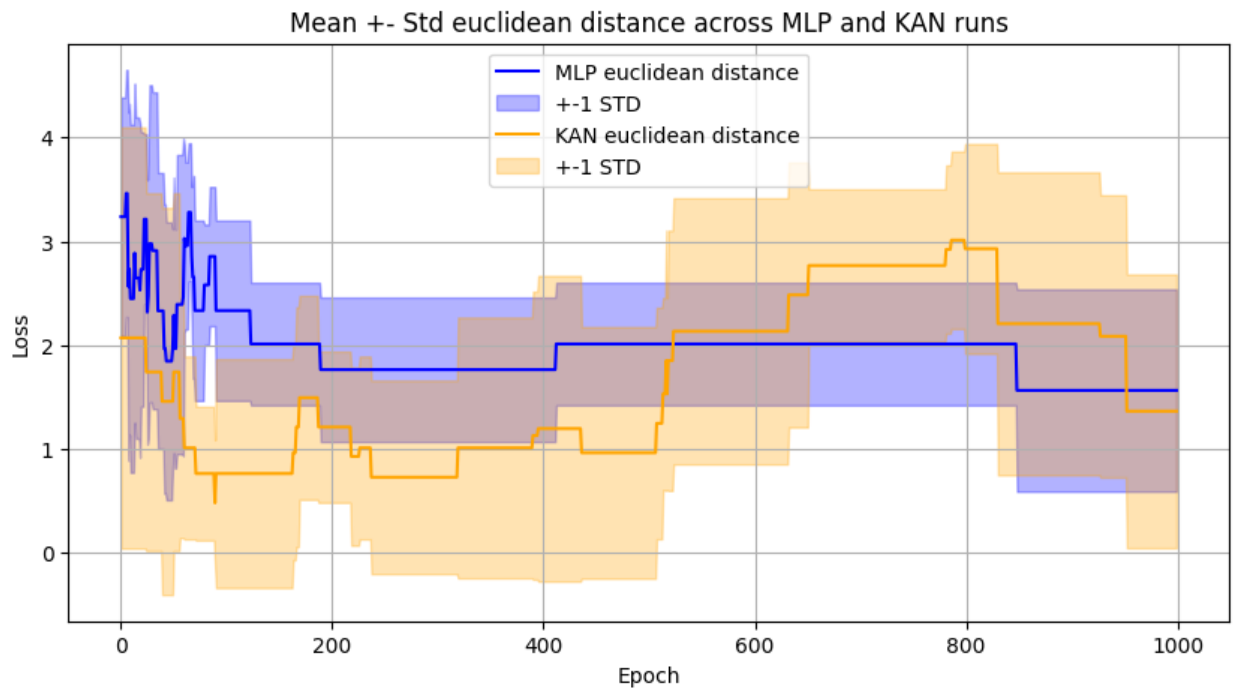


Figure 7-10: L2 norm over 1000 epochs for both models.

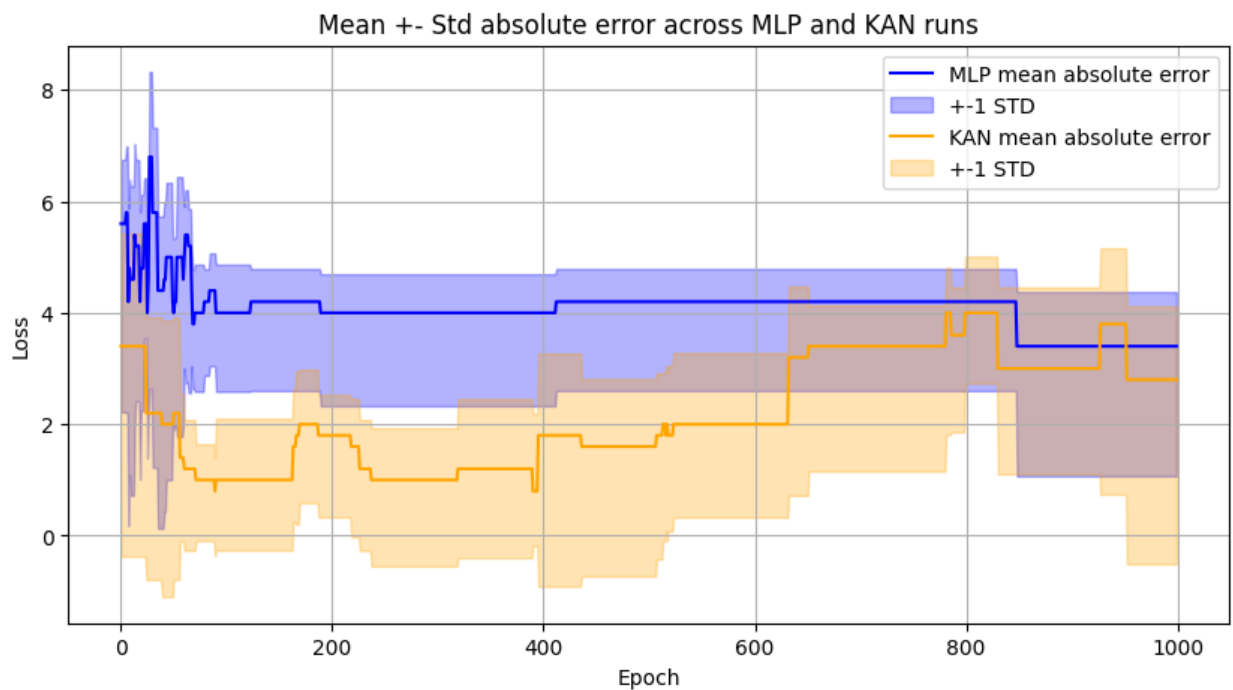


Figure 7-11: Absolute error over 1000 epochs for both models.



### Training time and convergence speed

The KAN was considerably faster on average, completing its training runs in about 45.6 seconds compared to 76.4 seconds for the MLP. It also demonstrated more consistent timing, with a lower standard deviation in training time (5.68 seconds for KAN vs 13.38 seconds for MLP). However, this large training time was due to a large amount of epochs being used for both models. In reality, neither model needed thousands of epochs to converge, as shown in figure 7-12.

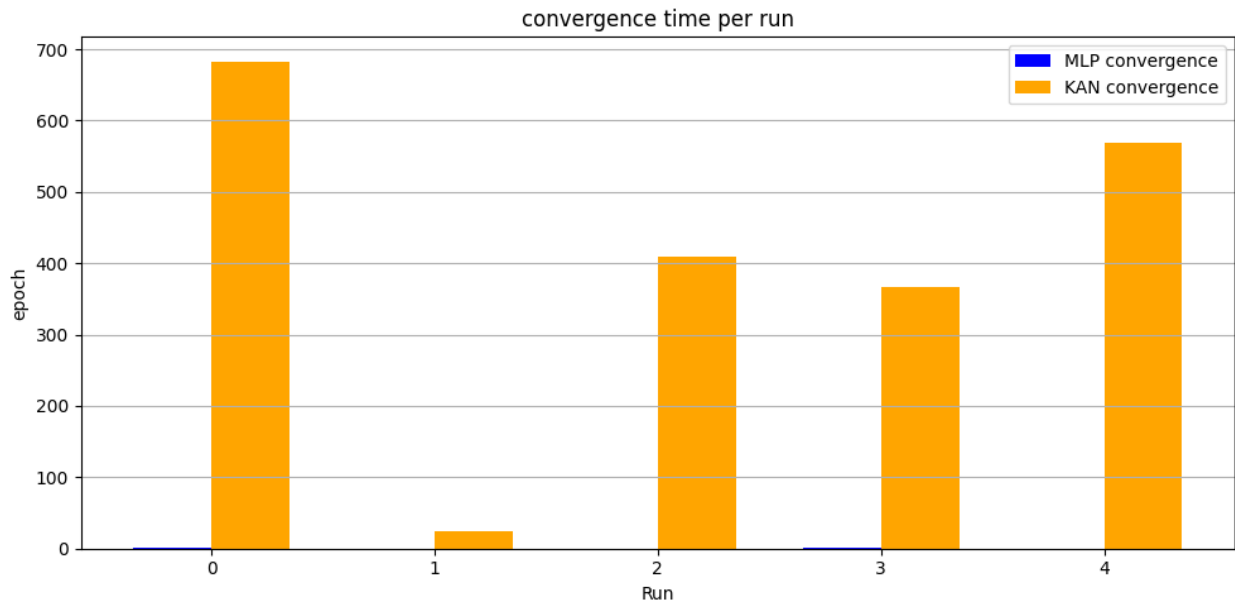


Figure 7-12: Graph of convergence times across all runs for both models. The MLP averaged just 2 epochs to converge.

The convergence behaviour further highlighted the difference between the two models. The MLP effectively converged almost immediately, with a mean convergence epoch of just 2, suggesting that it reached a stable output very early in training. Meanwhile, the KAN took around 532 epochs on average to stabilise, indicating a more gradual learning process. Although slower, this indicates that the KAN was able to explore the solution space more thoroughly, albeit unnecessarily.

### Model complexity

The KAN was significantly smaller in terms of trainable parameters, with only 360 parameters compared to the MLP's 1185, showing the KAN's efficiency in representing solutions with fewer resources.

### Summary

Although both models could find the global optimum in some cases, the MLP's faster convergence and lower training loss did not necessarily result in better objective performance. The KAN, despite slower convergence and higher loss values, managed to achieve slightly

higher mean objective values, showing better exploration of the solution space. However, given the simplicity of the problem, the MLP's rapid convergence and greater consistency suggest that, for small-scale integer problems like this, traditional MLPs are more practical.

## Problem 2 – Higher Dimensional

### Description

This second integer problem slightly increases the complexity by introducing an additional decision variable and more involved constraints. By increasing the dimensions, the search space increases massively. The objective is to maximise subject to constraints and bounded integer domain:

$$f(x_1, x_2, x_3) = 3x_1 + 5x_2 + 2x_3 \quad (7.2)$$

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &\leq 12 \\ x_1 + 2x_2 + 2x_3 &\leq 10 \\ 3x_1 + x_2 + 2x_3 &\leq 8 \\ 0 \leq x_1, x_2, x_3 &\leq 10, \quad x_1, x_2, x_3 \in \mathbb{Z} \end{aligned}$$

This scenario could again be interpreted as a discrete resource allocation problem with multiple constraints, where each variable represents an integer quantity of a product or action and the constraints encode budget or capacity limitations.

- **MLP:**
  - Architecture: 3 input neurons, two hidden layers of 32 neurons (ReLU activations), and a single linear output neuron.
  - Loss: Combines mean squared error between predicted and true objective values with a feasibility penalty that penalises constraint violations.
  - Training: AdamW optimiser with MSE loss, using soft penalties to encourage integer-like outputs. A learning rate of 0.001.
  - Data: Trained on 1000 feasible integer samples generated via rejection sampling.
- **KAN:**
  - Architecture: Width [3, 6, 6, 1] with B-spline activations and an initial grid resolution of 5.
  - Training: AdamW optimiser with loss function similar to MLP, no grid refinement applied.
  - Data: Also trained on 1000 integer-feasible points to respect the discrete problem structure.

The best solutions found are visualised below:

	X1	X2	X3	Objective
<b>MLP</b>	1	3	1	20
<b>KAN</b>	1	3	1	20
<b>Analytical</b>	1	3	1	20

Table 7-3: Summary of final results reached. Both models reach the exact optimal solution.

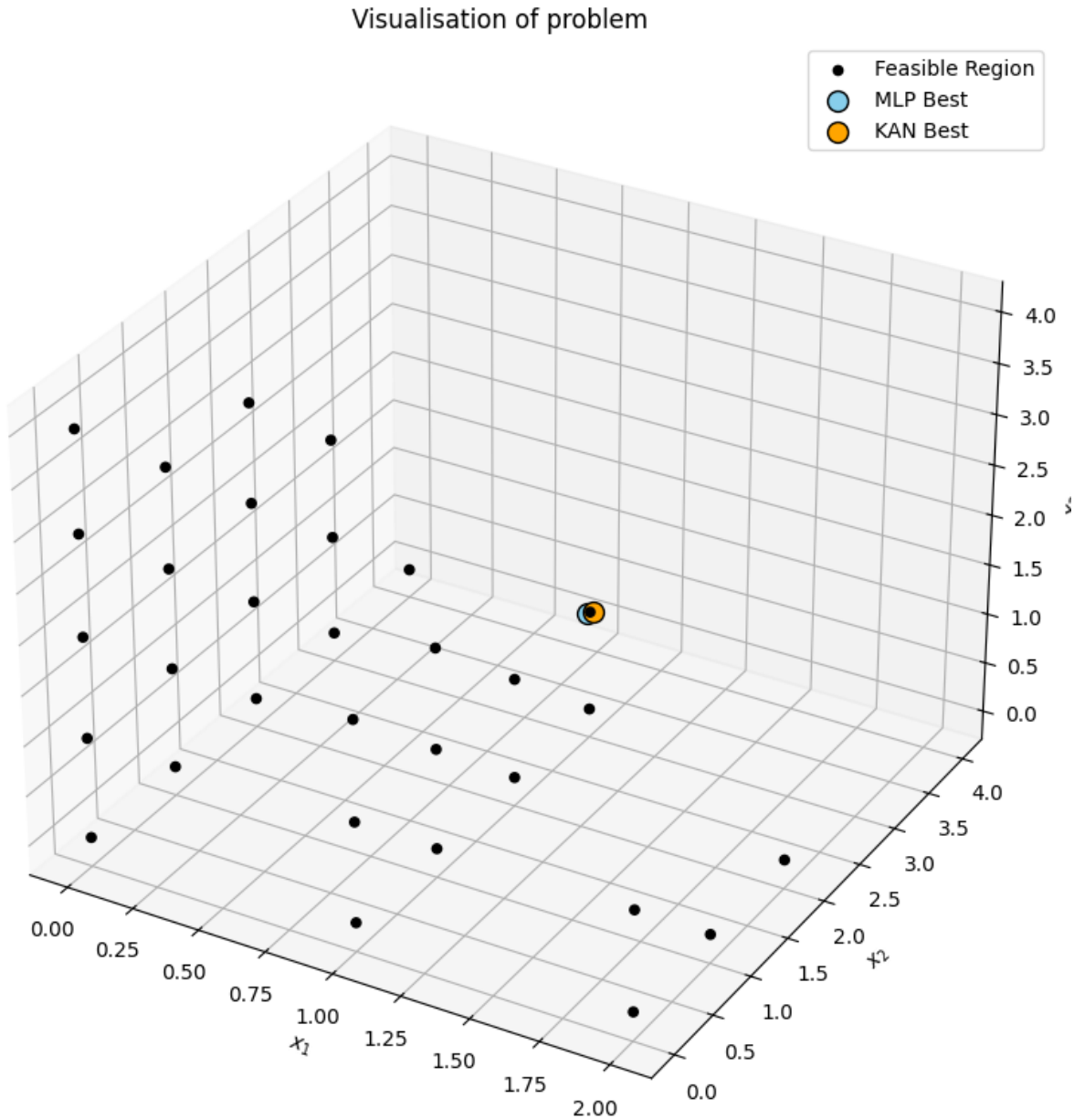


Figure 7-13: Visualisation of the solutions reached by each model.

## Analysis

The increase in dimensionality was able to provide some key insights into how the models performed.

Metric	MLP	KAN
Mean Objective Value	19.800000	19.600000
Std Objective Value	0.400000	0.8000000
Best Objective Value	20.000000	20.000000
Mean Final Loss	-0.002476	-0.002499
Std Final Loss	0.000039	0.0000063
Mean Time (s)	60.673300	79.88872
Std Time (s)	4.664675	18.26244
Mean Convergence Epoch	149.600000	236.8000
Std Convergence Epoch	23.938254	20.82691
Model Parameters	1217.0000	1192.0000

Table 7-4: Metrics comparison table.

### Objective value performance

Both the MLP and KAN models performed strongly on this integer-constrained optimisation task, with each successfully finding the global optimum of 20 in the majority of runs. On average, the MLP slightly outperformed the KAN, achieving a higher mean objective value of 19.8 compared to the KAN's 19.6. The MLP also showed more consistent results, reflected in its lower standard deviation in objective value of 0.4 vs 0.8, showing that it was generally more reliable across different runs.

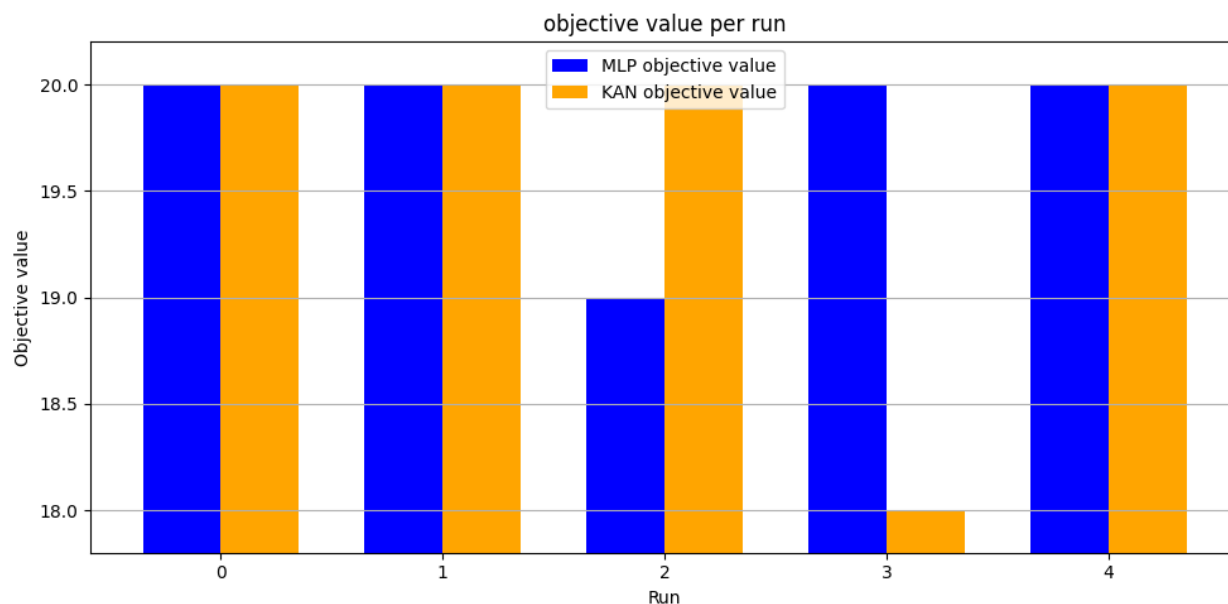


Figure 7-14: Objective values reached in each of the 5 runs for each model. Both reach the optimal value in 4 out of 5 runs.

## Loss and convergence behaviour

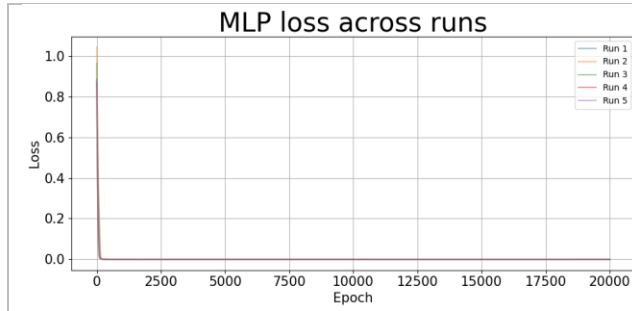


Figure 7-15: MLP loss curve over 20000 epochs.

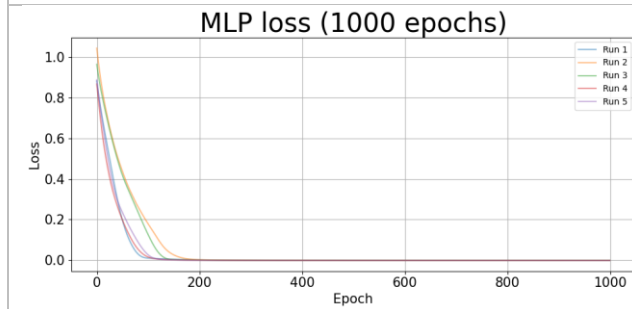


Figure 7-16: MLP loss curve over 1000 epochs.

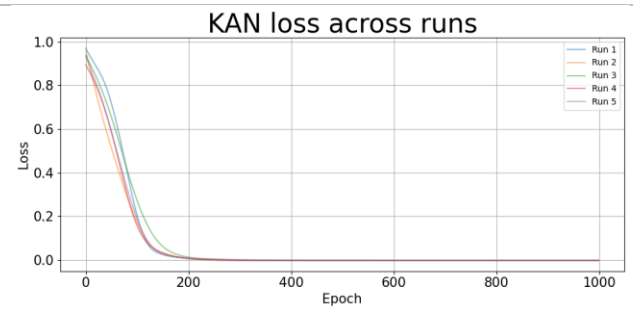


Figure 7-17: KAN loss curve over 1000 epochs.

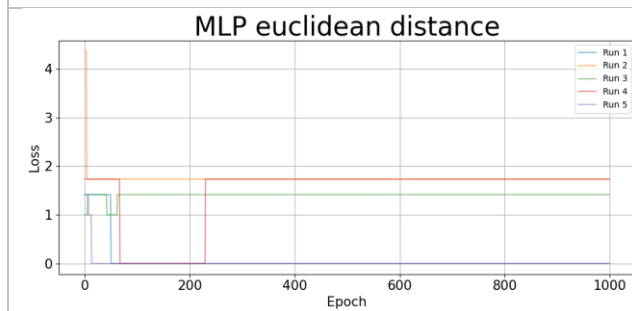


Figure 7-18: MLP L2 norm over 1000 epochs.

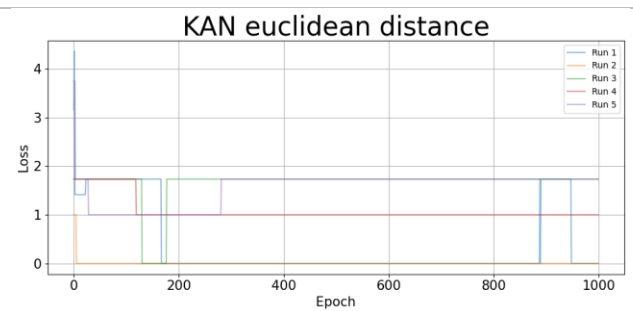


Figure 7-19: KAN L2 norm over 1000 epochs.

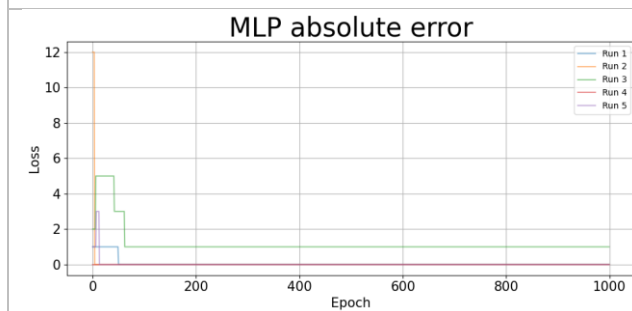


Figure 7-20: MLP absolute error over 1000 epochs.

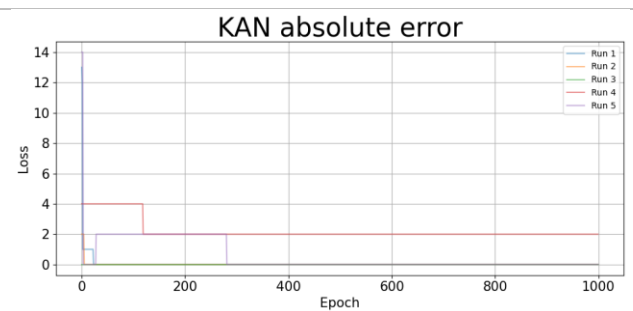


Figure 7-21: KAN absolute error over 1000 epochs.

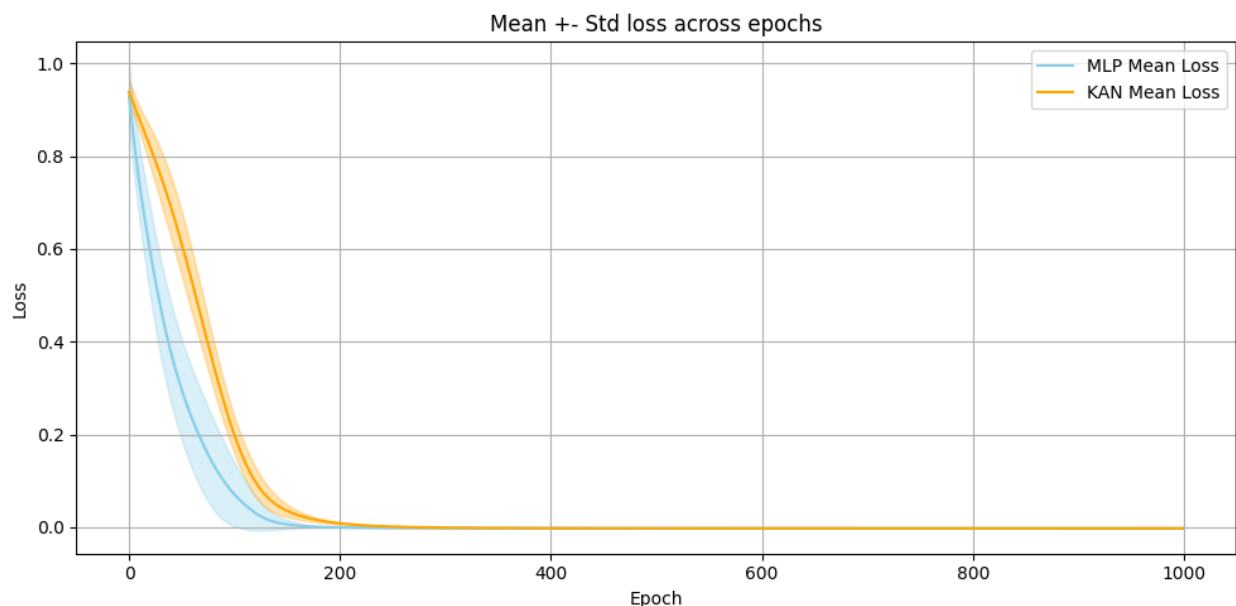


Figure 7-22: Loss curves of both models overlayed.

Both models achieved very low final losses, but the KAN obtained a slightly lower mean final loss (-0.002499) compared to the MLP (-0.002476). However, the differences were marginal, and the standard deviation of final losses was also extremely small for both. This indicates that both models were able to closely fit the data, with little variation between runs.

## Error observations

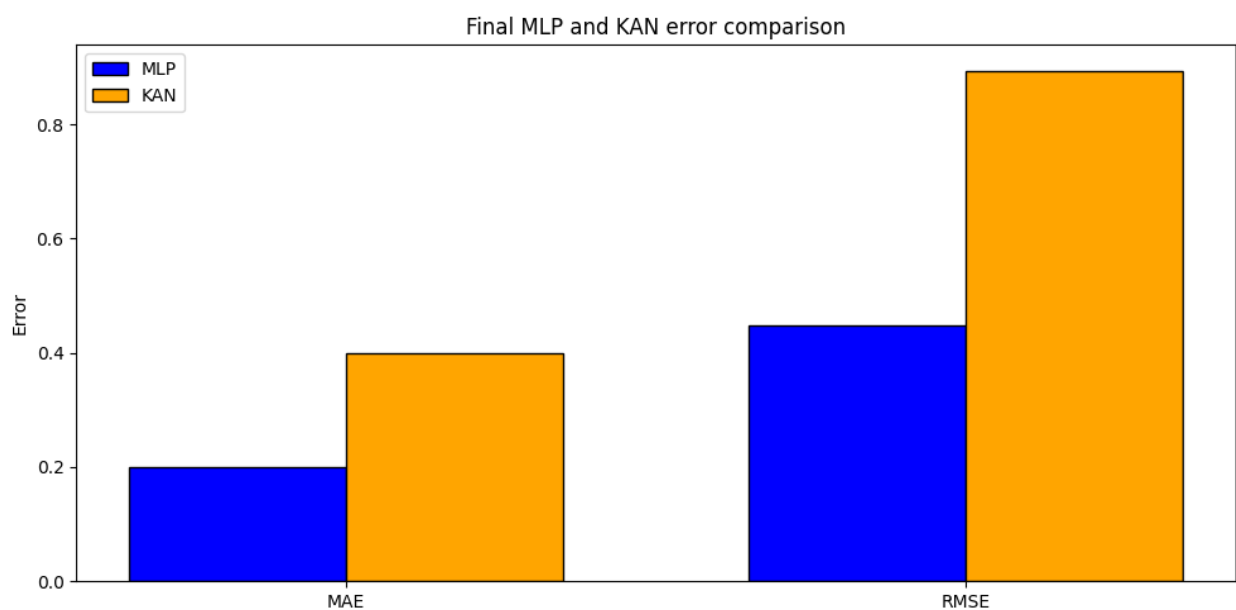


Figure 7-23: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.

The MLP clearly shows a lower MAE and RMSE error.

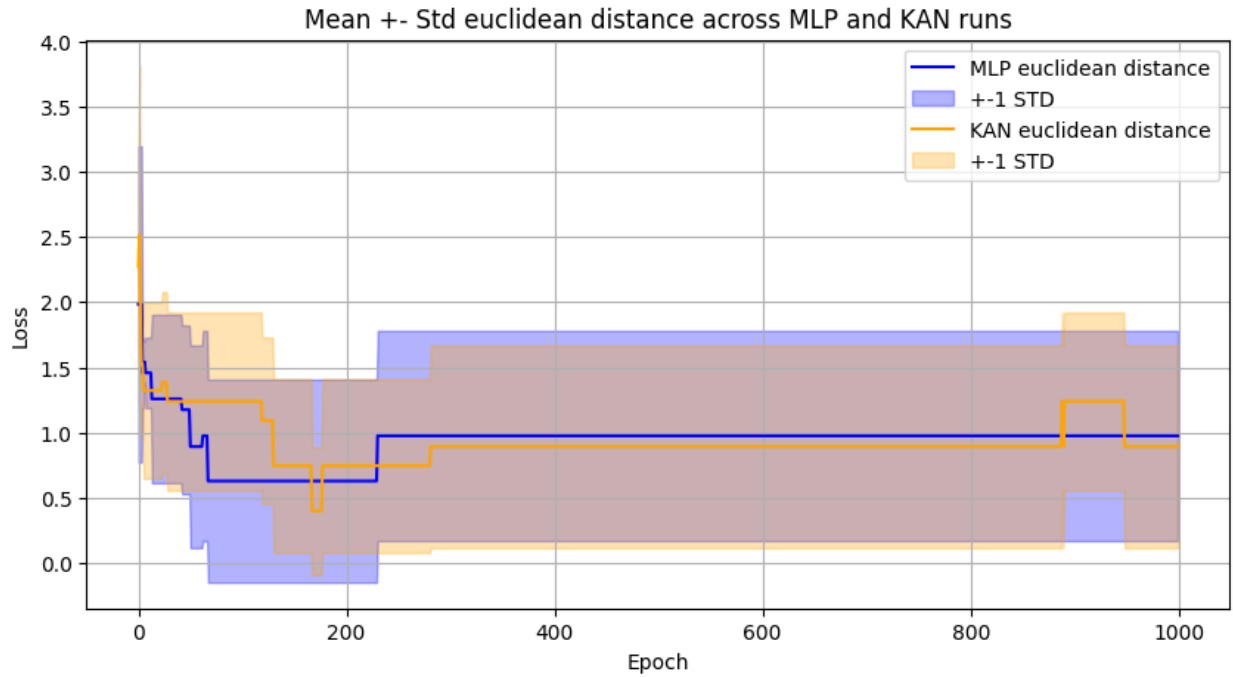


Figure 7-24: L2 norm over 1000 epochs for both models.

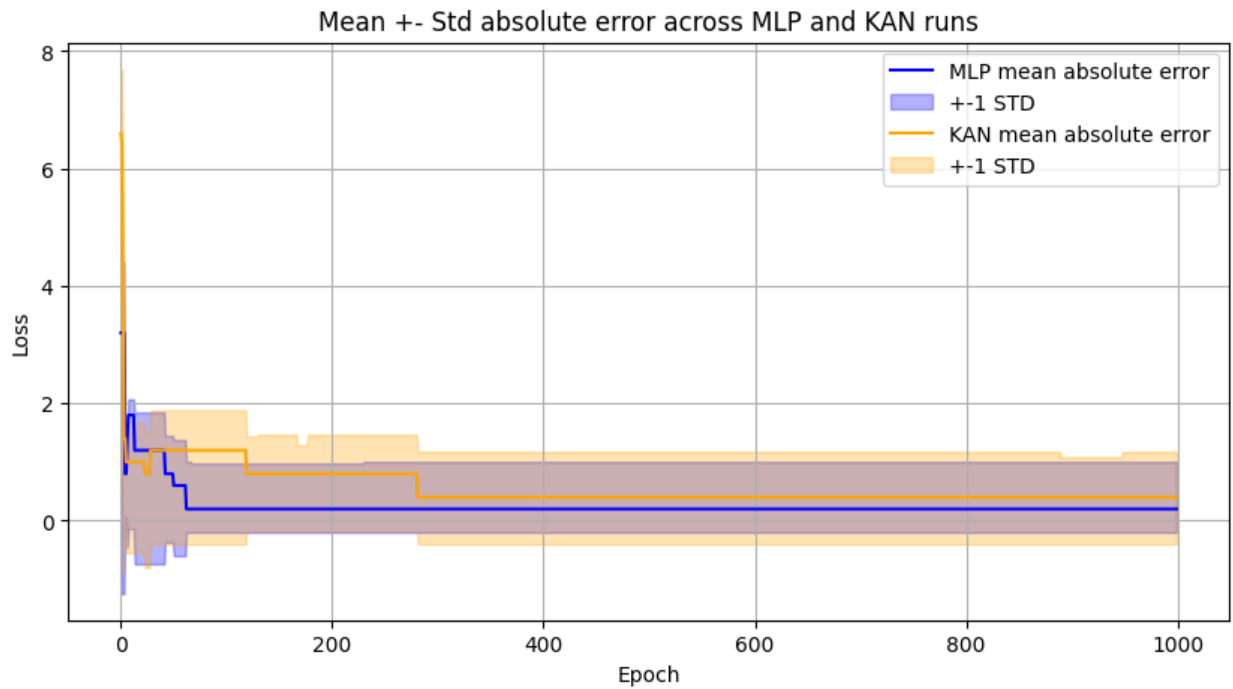


Figure 7-25: Absolute error over 1000 epochs for both models.

In both the L2 norm and absolute error observations, the MLP converged to a lower loss overall,

although it is interesting to see that the KAN had a larger spread in standard deviation, again indicating that it explores the search space better.

### Training time and model complexity

When it comes to training speed, the MLP was noticeably faster. It achieved convergence in an average time of around 60.7 seconds, compared to 79.9 seconds for the KAN. The larger variation in KAN training times, with a standard deviation of 18.3s, suggests that its performance was more sensitive to the initial conditions or the added complexity of grid refinement during training.

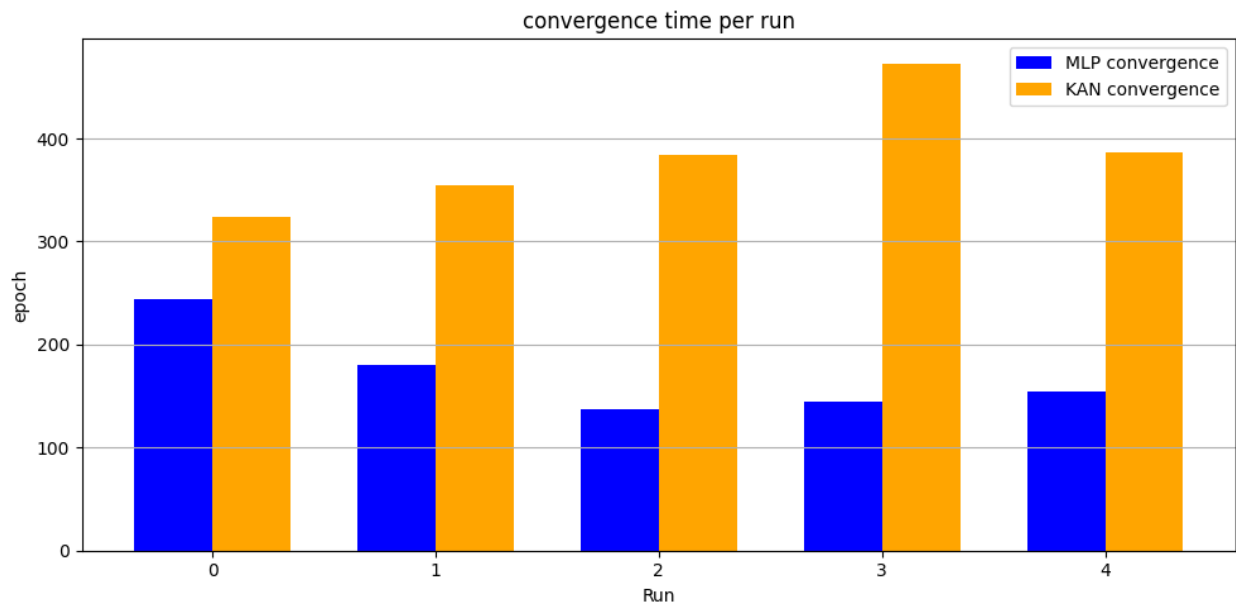


Figure 7-26: Graph of convergence times across all runs for both models.

The MLP also converged more quickly in terms of epochs. On average, it required about 150 epochs to converge, whereas the KAN needed roughly 237 epochs. This difference highlights that the MLP adapted faster to the relatively simple structure of the problem.

In terms of model complexity, both had similar parameter counts, at 1217 for the MLP and 1192 for the KAN, meaning that the differences in performance can mostly be attributed to the models' training dynamics rather than to architectural size or capacity.

### Summary

For this particular problem, the MLP is shown to be somewhat more suitable. It achieved a higher mean objective value, converged faster, and required less training time compared to the KAN. Both models were able to find the global optimum in at least one run, but the MLP demonstrated more consistent performance across runs. Although the KAN exhibited slightly



lower variability in its training losses, this did not translate into better final objective values. Given that the two models had similar amounts of parameters, it seems that the MLP’s simpler structure was better suited for this particular problem.

## Cross-problem Comparison

Overall, the MLP model seemed to outperform the KAN consistently across both problems. Although the MLP ran for significantly longer in terms of epochs, this was not strictly necessary, as the MLP found it easy to reach near-optimal or optimal solutions within a small number of epochs. In contrast, the KAN consistently required more training to achieve comparable solutions and showed slower convergence behaviour.

In problem 1, the MLP was able to converge quickly and maintain very low loss values, achieving near-optimal objectives across all runs with very little variability. The KAN, while able to reach good solutions, exhibited higher variance across runs and was slightly less consistent in hitting the exact optimal objective. The gap between the models widened in problem 2, where the problem’s search space increased massively. The MLP continued to converge rapidly and achieve higher mean objective values, while the KAN struggled to match this level of performance, despite using fewer parameters.

Across both problems, the discrete nature of integer optimisation appeared to favour the more direct structure of MLPs. The spline-based interpolation approach of the KAN seemed to not be as naturally suited to enforcing hard integer constraints, which may explain the relative underperformance. Nevertheless, the KAN was still able to remain competitive, which is promising, and implies that with further tuning or more specialised adaptations, KANs could become stronger candidates for discrete optimisation tasks in the future.

# Non-Convex Optimisation

## Problem 1 – Lower Dimensional

### Description

Non-convex problems are known to be more difficult due to the presence of multiple local minima and the lack of a globally smooth objective surface. The objective function is a mix of trigonometric and quadratic components, creating a non-convex surface with multiple basins. The problem formulation is as follows:

$$f(x, y) = \sin(3x) + \cos(5y) + x^2 + y^2 \quad (8.1)$$

$$\begin{aligned} x^2 + y^2 &\leq 4 \\ x - y &\geq -1 \\ -2 &\leq x \leq 2, \quad -2 \leq y \leq 2 \end{aligned}$$

This formulation simulates an environment where solutions must lie within a bounded region that combines both nonlinear and linear feasibility conditions. The objective is to find a combination of  $x$  and  $y$  that minimises the cost function while remaining within the feasible region.

The architecture of the models is as follows:

- **MLP:**
  - Architecture: 2 input neurons, two hidden layers with 64 neurons each (ReLU activation), 1 output neuron. A learning rate of 0.001 is used.
  - Loss: MSE between predicted and true objective values.
  - Training: Adam optimiser with standardised targets, trained for 20,000 epochs on 5000 feasible samples.
- **KAN:**
  - Architecture: Grid-based KAN with B-spline activations, width [2, 4, 4, 1], initial grid resolution 10.
  - Loss: MSE supervised learning, trained on the same feasible dataset as the MLP.
  - Training: Incorporates grid refinement at fixed epochs, increasing to 20 at 500 epochs.

This problem is a valuable benchmark for evaluating how well the models can learn non-convex objective landscapes, enforce constraints, and identify near-optimal feasible solutions.

The best solutions found are visualised below:

	<b>X1</b>	<b>X2</b>	<b>Objective</b>
<b>MLP</b>	-0.4660	-0.5639	-1.4118
<b>KAN</b>	-0.4453	-0.5849	-1.4109
<b>Analytical</b>	-0.42730784	-0.58137849	-1.4106

Table 8-1: Summary of final results reached. The KAN reaches a solution that is closer to the true solution than the MLP.

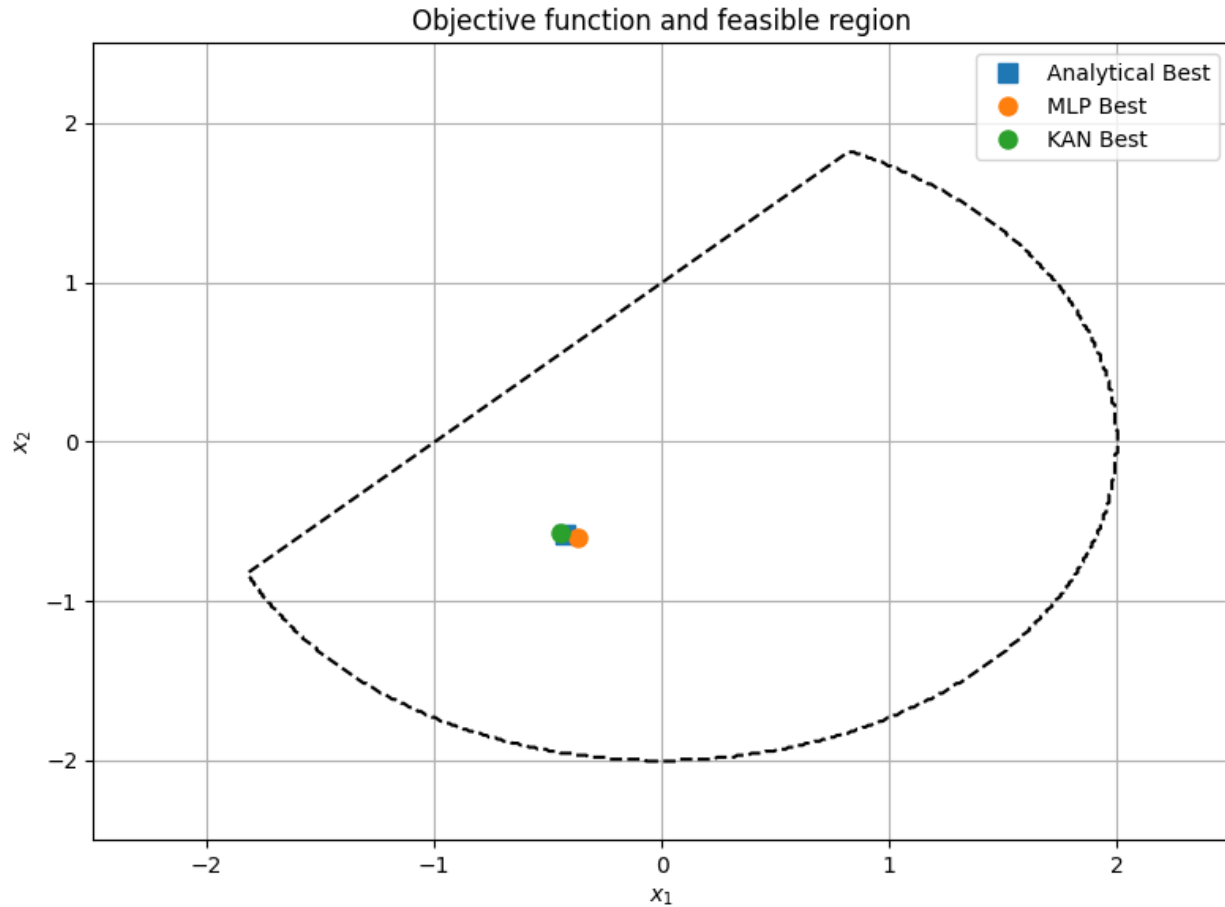


Figure 8-1: Visualisation of the solutions reached by each model. The KAN is shown to reach closer to the actual solution.

## Analysis

The introduction of a non-convex problem allowed the KAN's advantages to be seen in a much more visible way.

<b>Metric</b>	<b>MLP</b>	<b>KAN</b>
<b>Mean Objective Value</b>	-1.412978	-1.404441
<b>Std Objective Value</b>	0.012564	0.007957
<b>Best Objective Value</b>	-1.395410	-1.395774
<b>Mean Final Loss</b>	0.000641	0.004348
<b>Std Final Loss</b>	0.000506	0.000963
<b>Mean Time (s)</b>	164.020366	188.836015

<b>Std Time (s)</b>	32.815814	1.338182
<b>Mean Convergence Epoch</b>	16879.200000	750.600000
<b>Std Convergence Epoch</b>	3394.267308	184.781601
<b>Model Parameters</b>	4417.000000	794.000000

Table 8-2: Metrics comparison table

### Objective value performance

Both models performed well on this task, converging consistently toward near-optimal solutions. The objective value found by the analytical solver was -1.4106, and in regards to this, the MLP achieved a slightly worse mean objective value of -1.413 compared to -1.404 for the KAN. The KAN's lower standard deviation (0.00796 vs 0.01256) suggests slightly more consistent performance across different training runs.

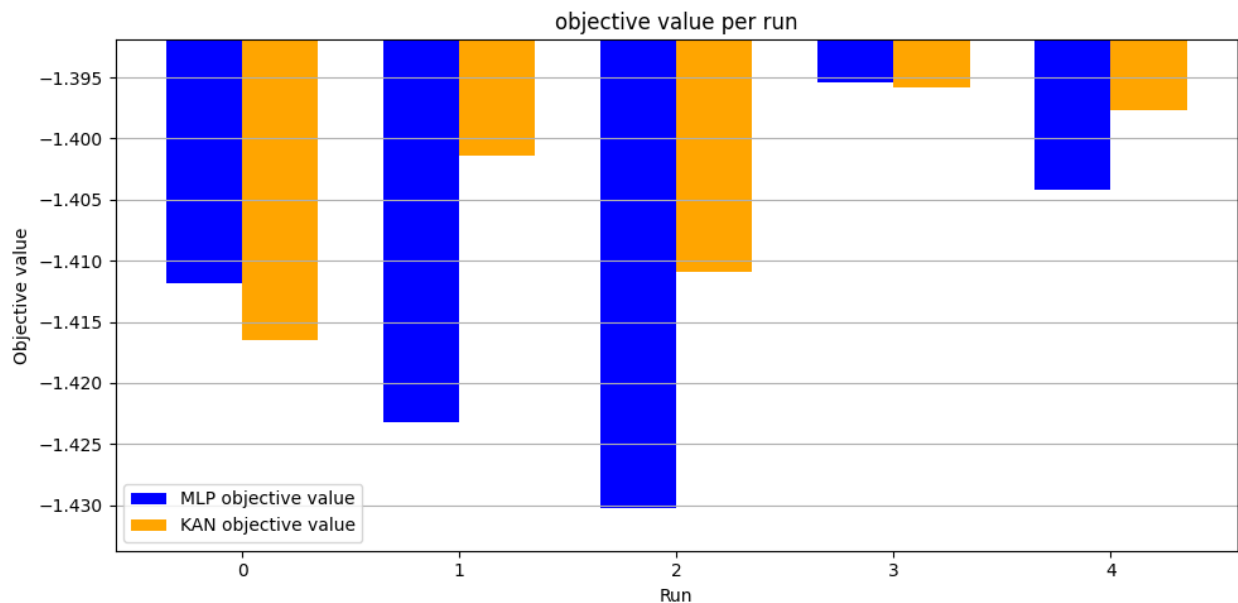


Figure 8-2: Objective values reached in each of the 5 runs for each model.

From the runs in figure 8-2, the MLP was able to reach more minimal objective values consistently, but they did not always satisfy the constraints. Taking that into consideration, the KAN was able to get closer to the actual optimal solution.

### Loss and convergence behaviour

The loss graphs from figures 8-3 to 8-9 consistently demonstrate that the KAN model is able to achieve a lower loss faster, whereas the MLP is slower to converge.

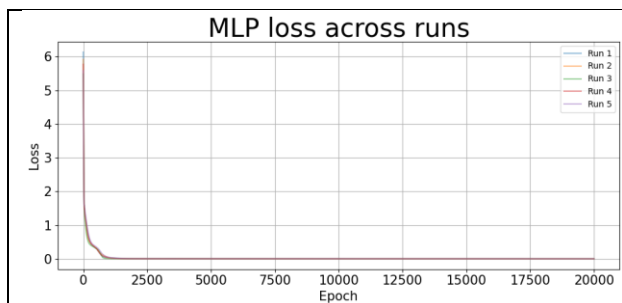


Figure 8-3: MLP loss curve over 20000 epochs.

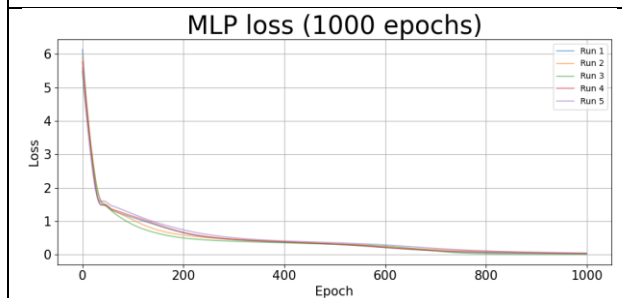


Figure 8-4: MLP loss curve over 1000 epochs.

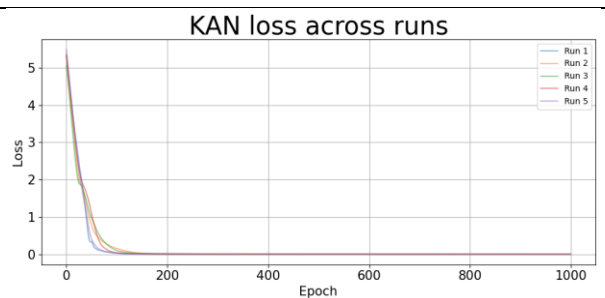


Figure 8-5: KAN loss curve over 1000 epochs.

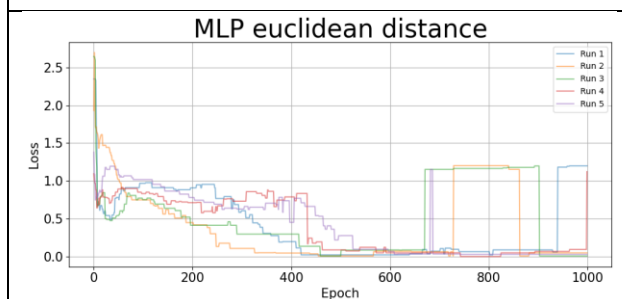


Figure 8-6: MLP L2 norm over 1000 epochs.

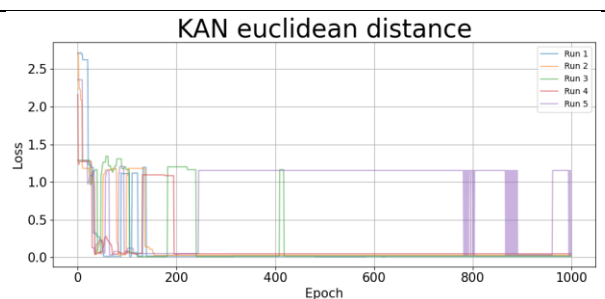


Figure 8-7: KAN L2 norm over 1000 epochs.

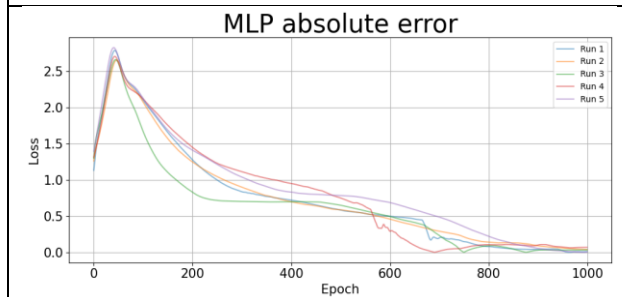


Figure 8-8: MLP absolute error over 1000 epochs.

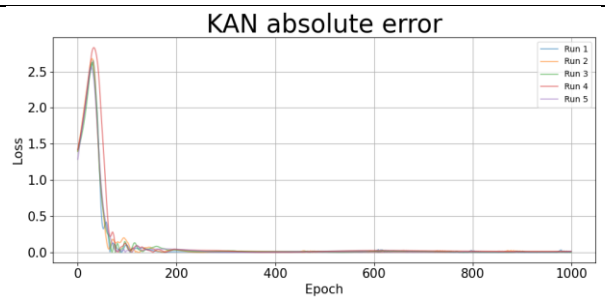


Figure 8-9: KAN absolute error over 1000 epochs.

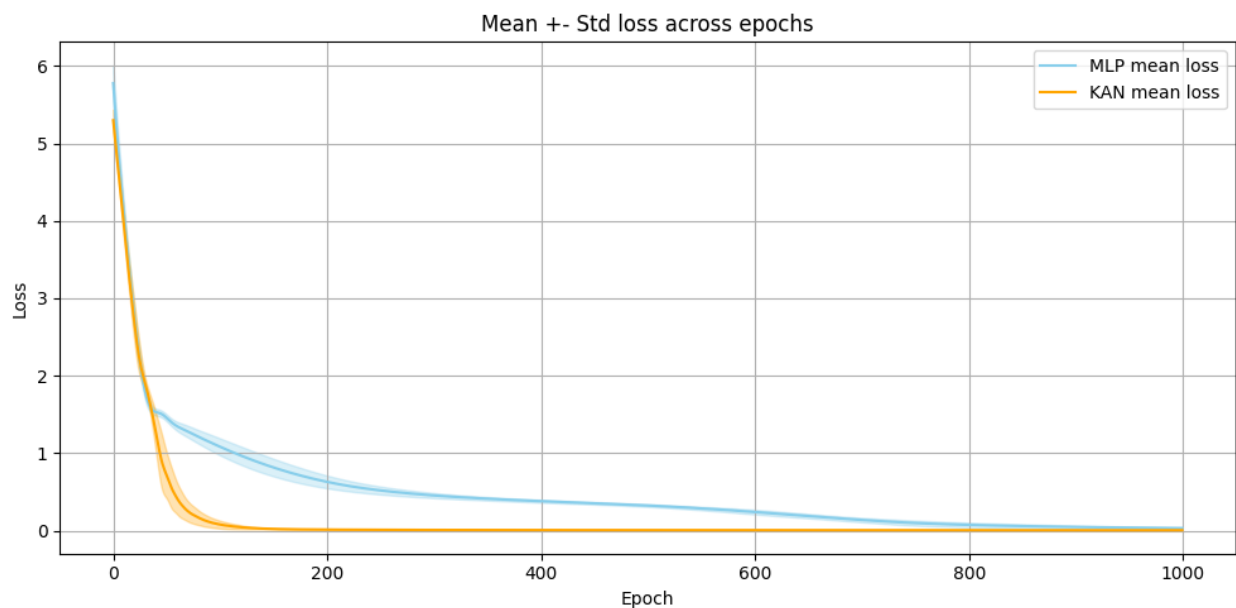


Figure 8-10: Loss curves of both models overlayed.

When comparing training loss, from figure 8-10 it is evident that the KAN was able to converge faster. The MLP seemed to struggle more than the KAN up to 900 epochs or so. However, the MLP had a slight edge, achieving a lower mean final loss of 0.000641, while the KAN settled at a higher 0.004348.

## Error observations

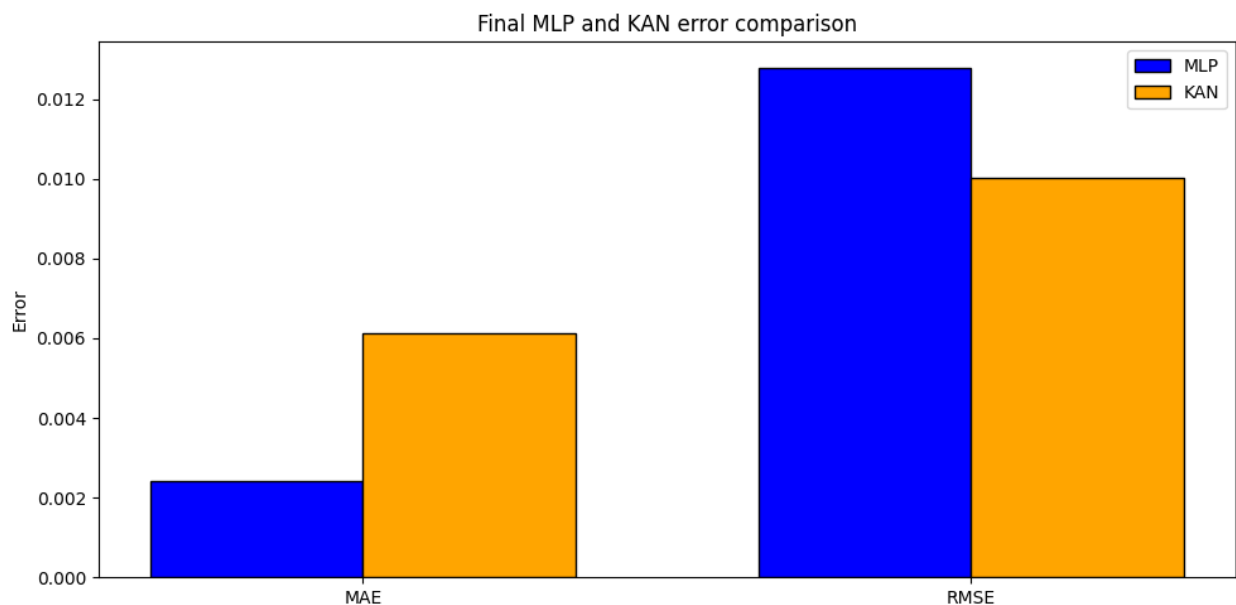


Figure 8-11: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.

Interestingly, figure 8-11 shows that the MLP has a smaller MAE whereas the KAN has a smaller RMSE. This could be because the MLP creates sharper piecewise linear approximations, resulting in a large number of jumps, causing larger errors. Figure 8-13, showing the graph of L2 norm, supports this. On the other hand, the KAN may produce smoother, more gradual approximations, leading to many small deviations that could increase the MAE.

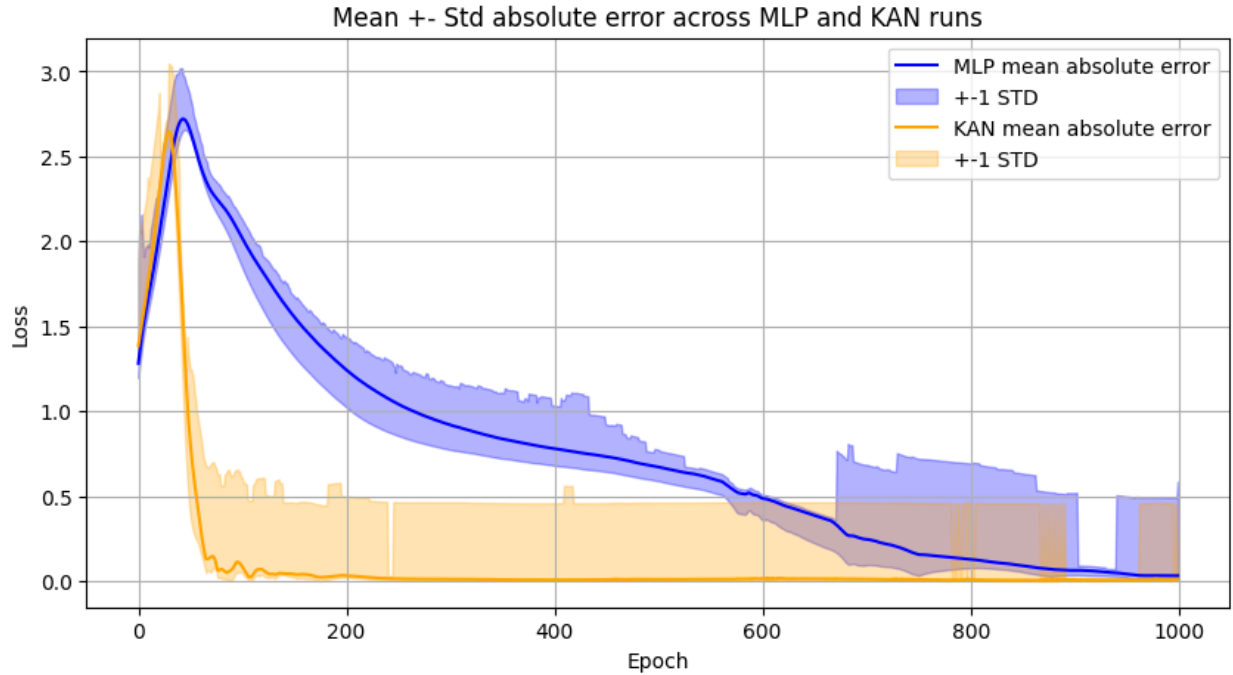


Figure 8-12: Absolute error over 1000 epochs for both models.

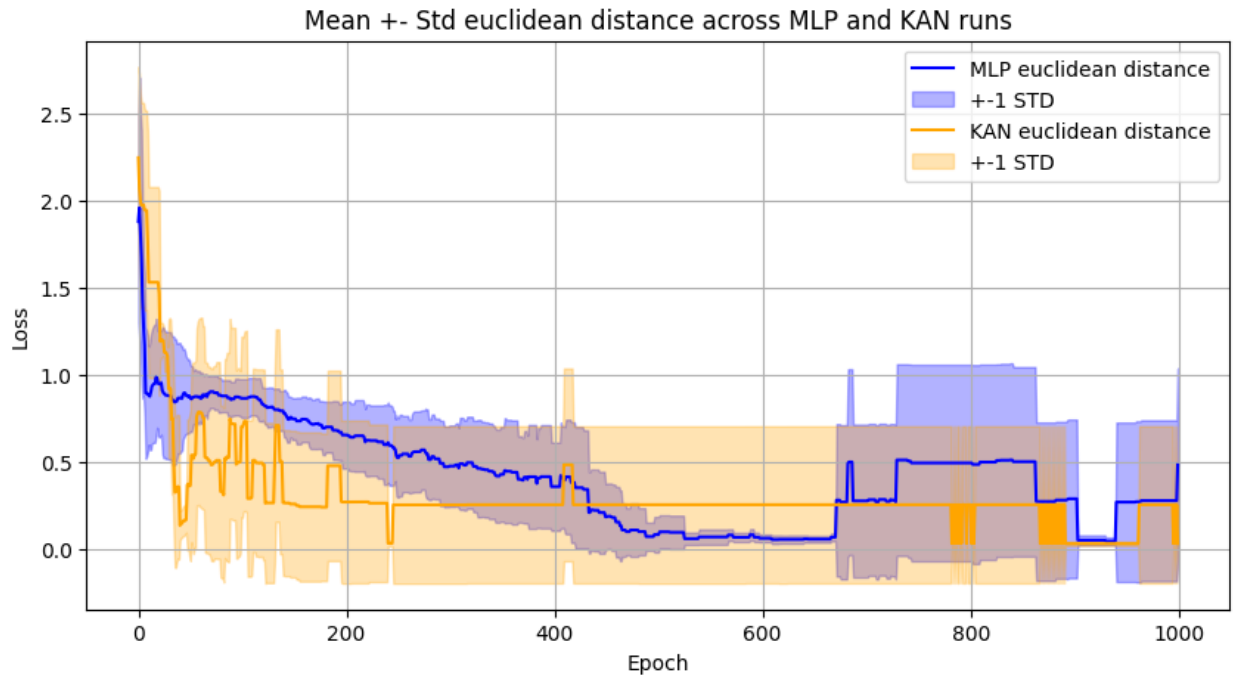


Figure 8-13: L2 norm over 1000 epochs for both models.

Based on the L2 norm and absolute error metrics shown in figures 8-12 and 8-13, the KAN consistently outperforms the MLP, reaching a low loss quickly. Interestingly, both models spike in absolute error towards the start, potentially indicating how the nature of the problem forces the models to initially make rougher, exploratory guesses.

### Training time and convergence time

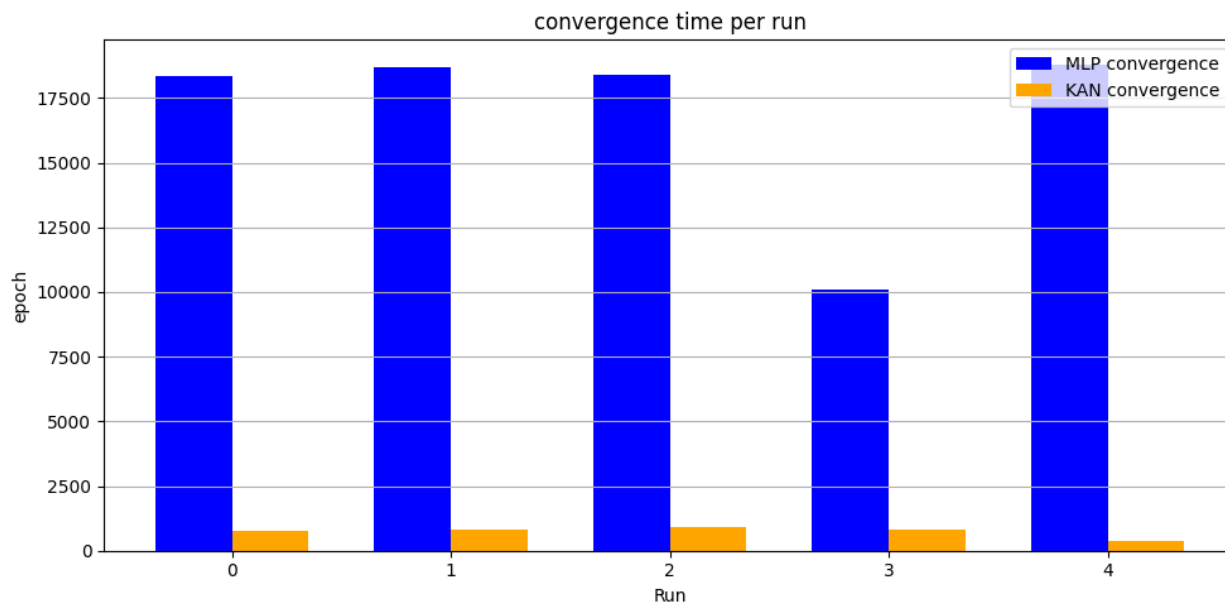


Figure 8-14: Graph of convergence times across all runs for both models.

In terms of runtime, the MLP completed training slightly faster on average, at 164 seconds per run vs 189 seconds for KAN. However, the KAN was far more consistent, with a standard deviation in training time of only 1.34 seconds compared to 32.8 seconds for the MLP, which could be useful for practical deployments where predictable performance is critical.

A particularly noticeable difference emerged in convergence speed, in figure 8-14. The KAN reached convergence in under 800 epochs on average, whereas the MLP required nearly 17,000 epochs to achieve similar stability. This again shows the KAN's efficiency in learning a reliable representation of the function much earlier than its counterpart, despite its higher per-epoch cost.

### Model complexity

The KAN was also much lighter in terms of model complexity. Using only 794 trainable parameters, it managed to perform better than the MLP, which required 4417 parameters to achieve near comparable performance. This reflects KAN's structural efficiency and its ability to represent complex functions with fewer resources.

### Summary

Overall, the KAN achieved slightly better objective values, with a fraction of the complexity. It



converged much faster, and more consistently. However, it required significantly more training time and achieved a slightly lower final loss, although it outperformed on the MAE and RMSE measurements. It is clear that it is a strong candidate for non-convex optimisation tasks.

## Problem 2 – 3D Rastrigin function

### Description

This problem is designed to evaluate how well models perform under multi-modal and high-dimensional non-convex optimisation conditions. Specifically it uses the Rastrigin function, which is a well-known benchmark function characterised by its periodic structure and large number of local minima. The function is defined as the following, subject to constraints:

$$\begin{aligned}
 \min f(x_1, x_2, x_3) & \quad (8.2) \\
 &= 30 + (x_1^2 - 10 \cos(2\pi x_1)) \\
 &\quad + (x_2^2 - 10 \cos(2\pi x_2)) \\
 &\quad + (x_3^2 - 10 \cos(2\pi x_3)) \\
 5.12 &\leq x_1, x_2, x_3 \leq 5.12 \\
 x_1 + x_2 + x_3 &\leq 10 \\
 x_1 + x_2 + x_3 &\geq 7 \\
 x_1 - x_2 &\geq -3
 \end{aligned}$$

Because of the many local minima, it is a challenging landscape for optimisation models, particularly those that rely on local gradients. The architectures are as follows:

- **MLP:**
  - Architecture: 3 input neurons, two hidden layers of 64 neurons each (ReLU activations), and one output neuron.
  - Loss function: Mean squared error (MSE) between predicted and true objective values.
  - Training strategy: Trained using Adam optimiser across 20000 epochs, using 5000 synthetic samples from the feasible domain.
- **KAN:**
  - Architecture: Width configuration of [3, 8, 8, 1] with grid resolution 50, using spline-based edge activations.
  - Loss function: MSE between KAN predictions and ground-truth objective values, with grid refinement applied to enhance local fitting.
  - Training strategy: Adam optimiser used for 2000 epochs, with a grid refinement to 100 at 1000 epochs.

The best solutions found are visualised below:

	<b>X1</b>	<b>X2</b>	<b>X3</b>	<b>Objective</b>
<b>MLP</b>	4.0647	2.0016	0.9743	27.9561
<b>KAN</b>	2.0346	0.9903	3.9973	25.0393
<b>Analytical</b>	2.9966389	2.9966389	1.0067222	18.9866

Table 8-3: Summary of final results reached. The KAN is shown to perform better.

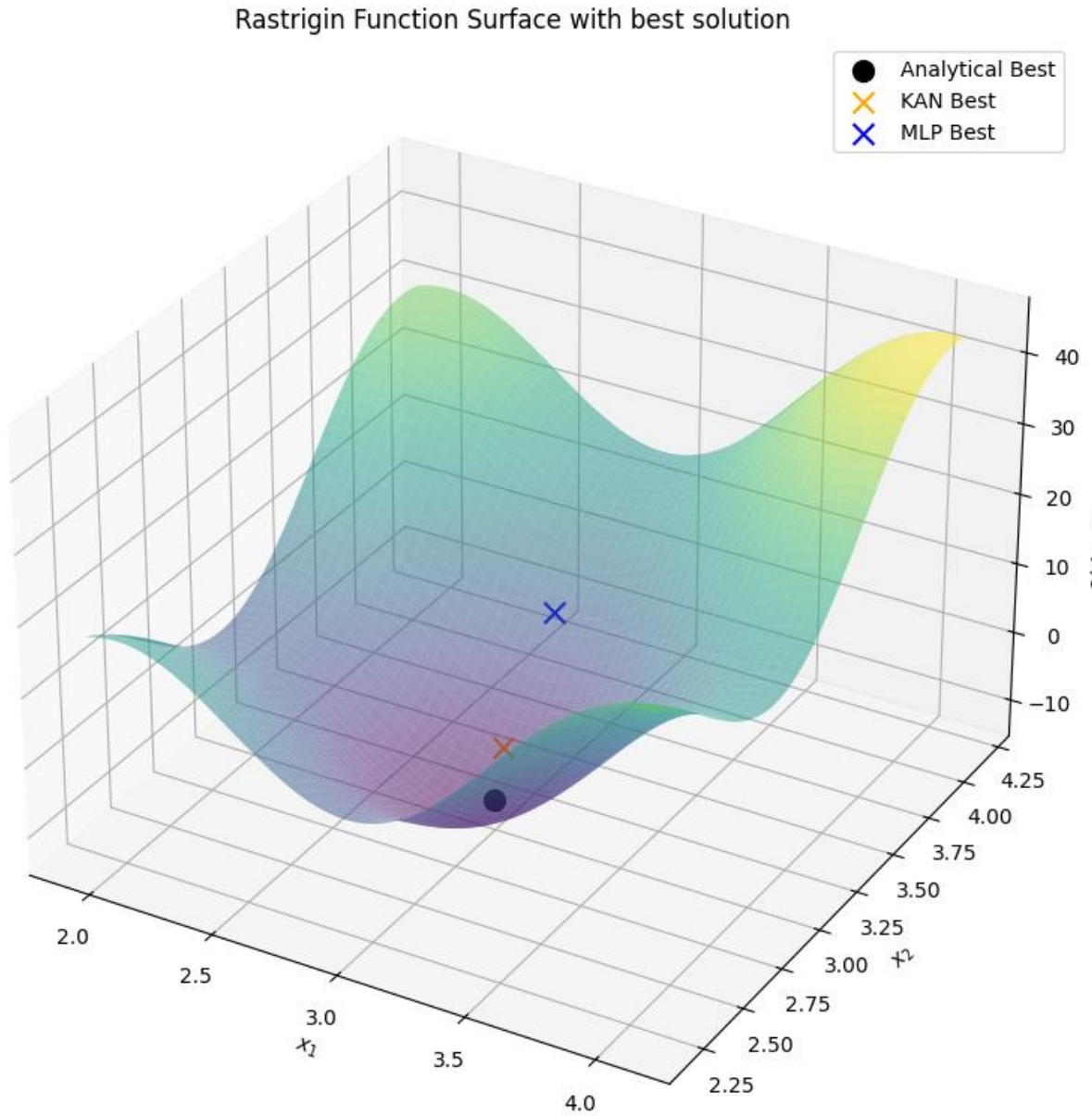


Figure 8-15: Visualisation of the solutions reached by each model. The KAN is shown to reach closer to the actual solution.

## Analysis

The KAN was able to demonstrate a significant advantage in performance compared to the MLP.

Metric	MLP	KAN
Mean Objective Value	32.522828	28.183926
Std Objective Value	3.811510	2.268204
Best Objective Value	27.9561	25.0393
Mean Final Loss	96.310957	52.848995
Std Final Loss	4.334031	14.869892
Mean Time (s)	60.756313	214.579485
Std Time (s)	0.678646	82.316157
Mean Convergence Epoch	14964.600000	1895.800000
Std Convergence Epoch	1345.393712	42.714869
Model Parameters	4481.000000	7007.000000

Table 8-4: Metrics comparison table

### Objective value performance

In this more challenging problem, both models struggled to reach the true global minimum, but the KAN model clearly outperformed the MLP in terms of minimising objective value. The MLP achieved a mean objective of 32.52, noticeably higher than the KAN's 28.18. The KAN also secured a better best-case solution across runs, implying that the KAN was more effective at navigating the rugged optimisation landscape. Also, it is worth noting that the MLP exhibited higher variability across runs, with a standard deviation of 3.81 compared to 2.27 for the KAN. This suggests that the KAN was more consistent overall.

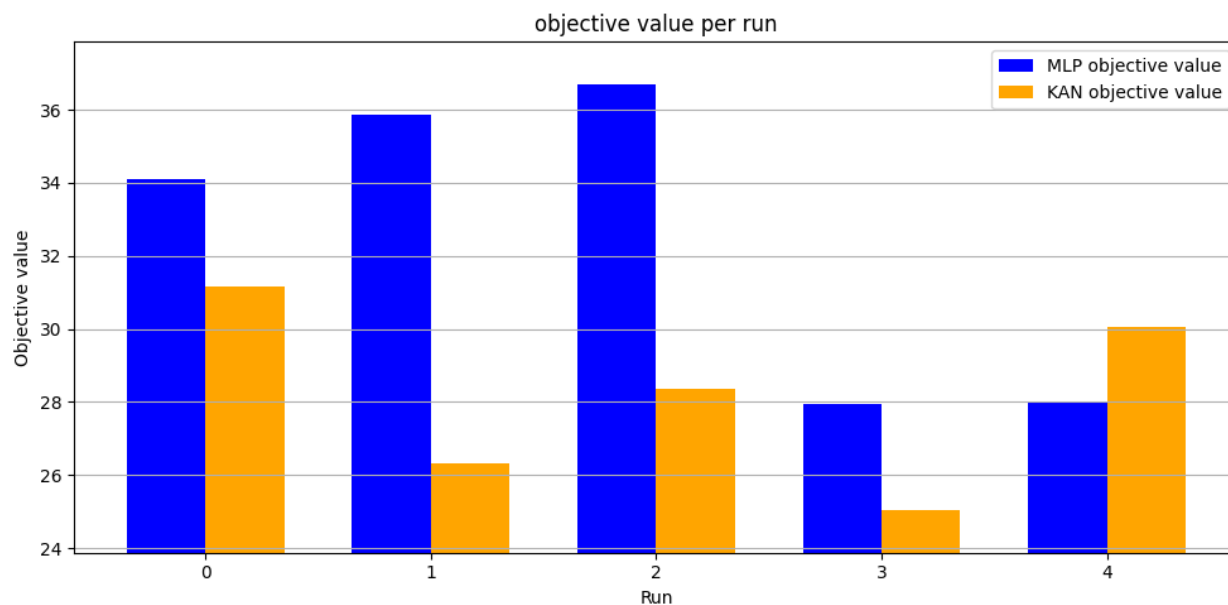


Figure 8-16: Objective values reached in each of the 5 runs for each model.

From figure 8-16, it is evident that the KAN consistently reached a lower objective value, although neither was able to consistently reach very close to the true optimal objective value.

### Loss and convergence behaviour

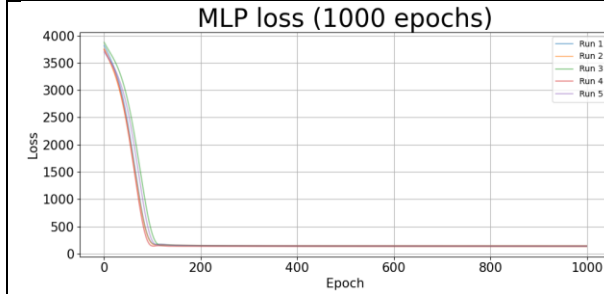


Figure 8-17: MLP loss curve over 1000 epochs.

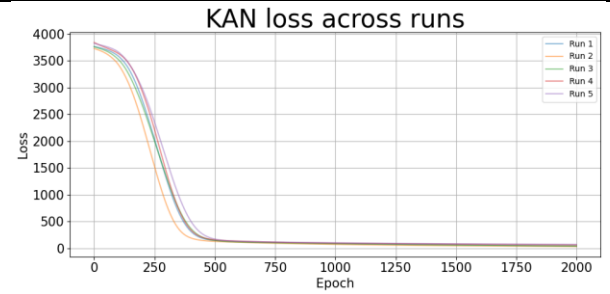


Figure 8-18: KAN loss curve over 2000 epochs.

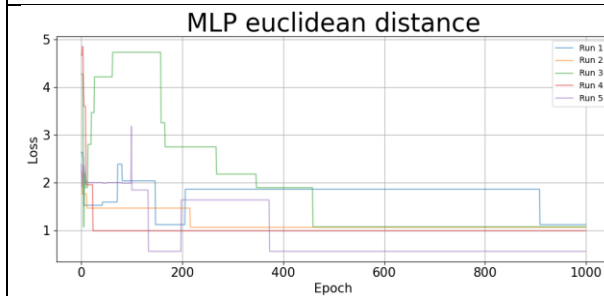


Figure 8-19: MLP L2 norm over 1000 epochs.

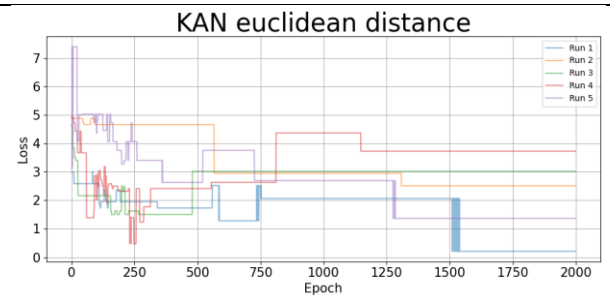


Figure 8-20: KAN L2 norm over 2000 epochs.

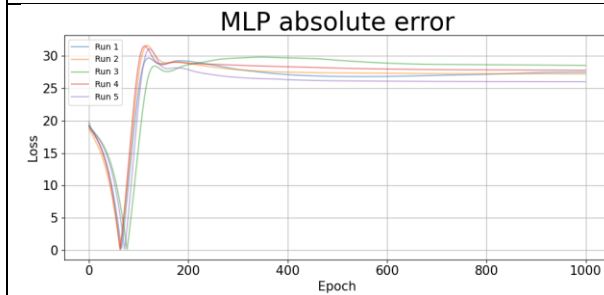


Figure 8-21: MLP absolute error over 1000 epochs.

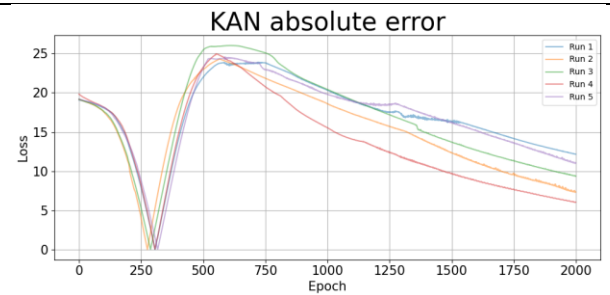


Figure 8-22: KAN absolute error over 2000 epochs.

Interestingly, The KAN showed much greater variability in final loss, with a standard deviation of 14.87 vs 4.33 for the MLP, highlighting its exploration effectiveness in more complex, irregular landscapes. The figure 8-21 supports this, showing how the KAN was able to search a wider variety of potentially optimal points, compared to the MLP in figure 8-20.

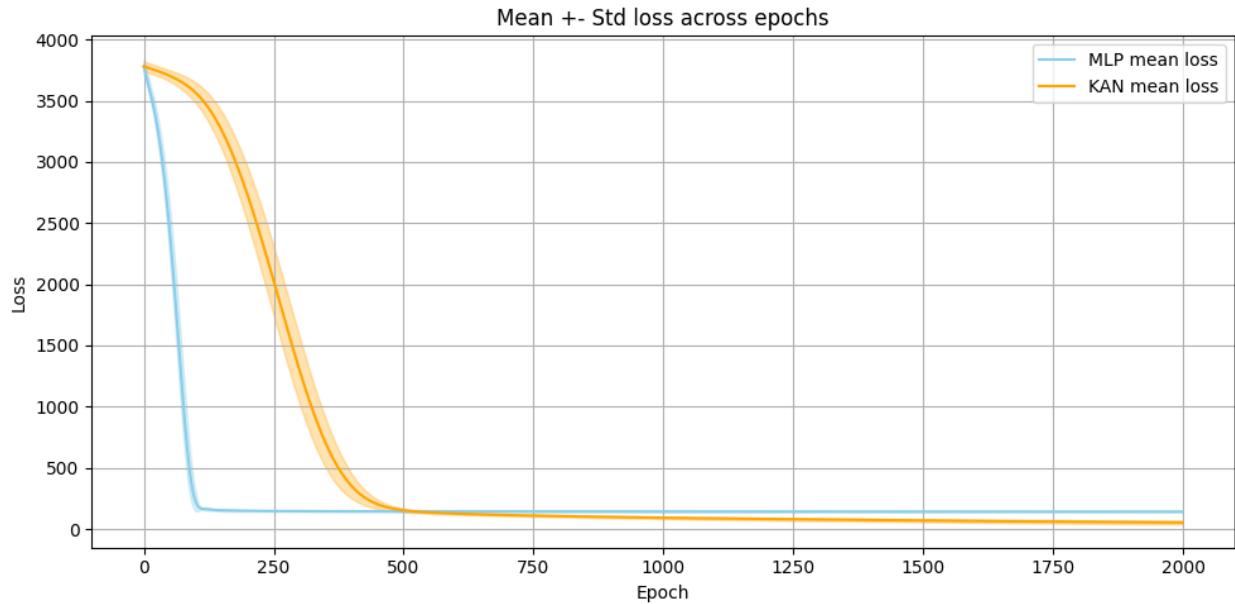


Figure 8-23: Loss curves of both models overlayed.

In figure 8-24, The MLP loss drops relatively quickly, whereas the KAN loss is more gradual, and consistently improves after overtaking the MLP at approximately 500 epochs. Given that the KAN generally reaches better objective values, this indicates that the MLP could have suffered from overfitting in this case.

### Error observations

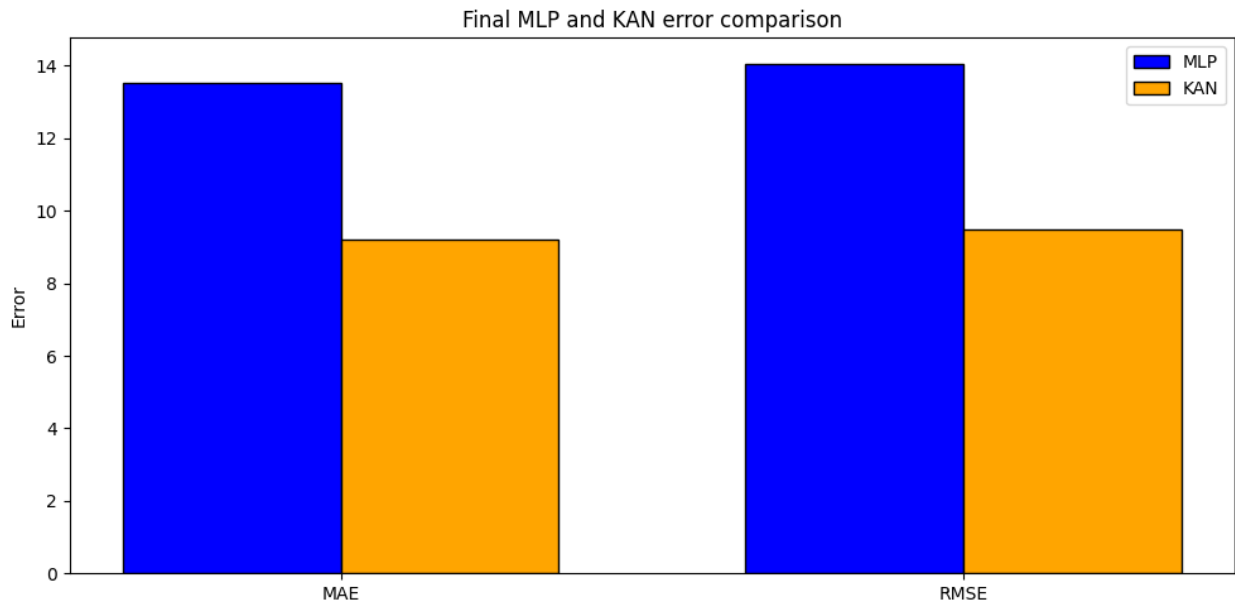


Figure 8-24: Visualisation of Mean Absolute Error and Relative Mean Squared Error for both models.

The error comparison graph above demonstrates the KAN's significantly better performance compared to the MLP, reaching much better MAE and RMSE values

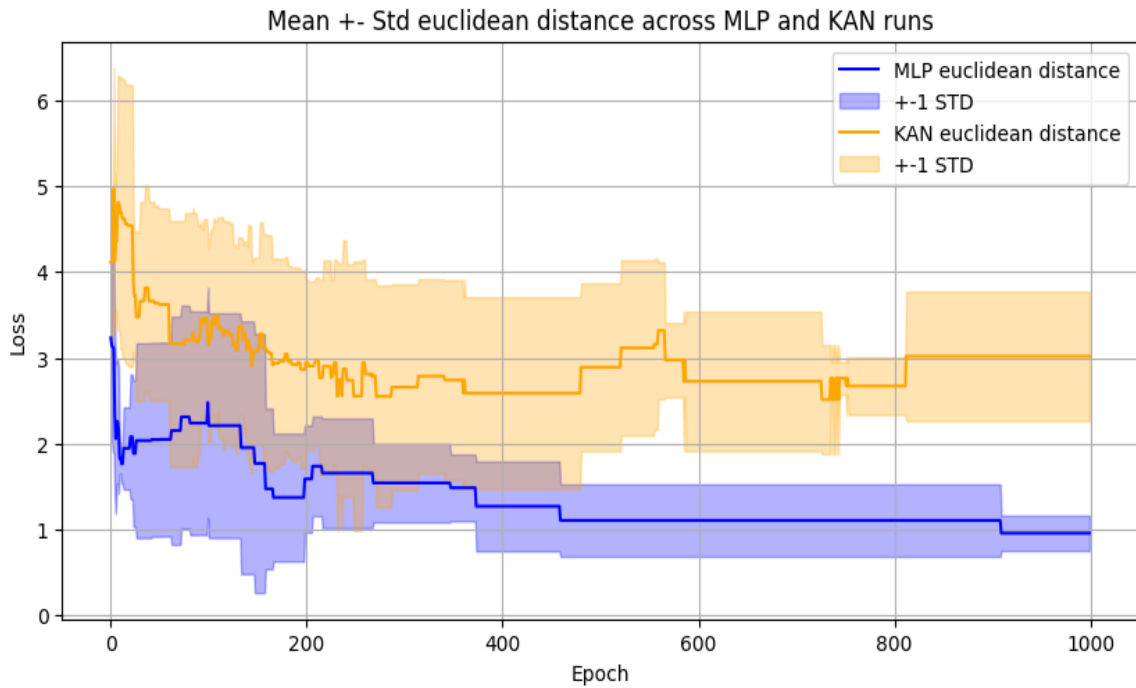


Figure 8-25: L2 norm over 1000 epochs for both models.

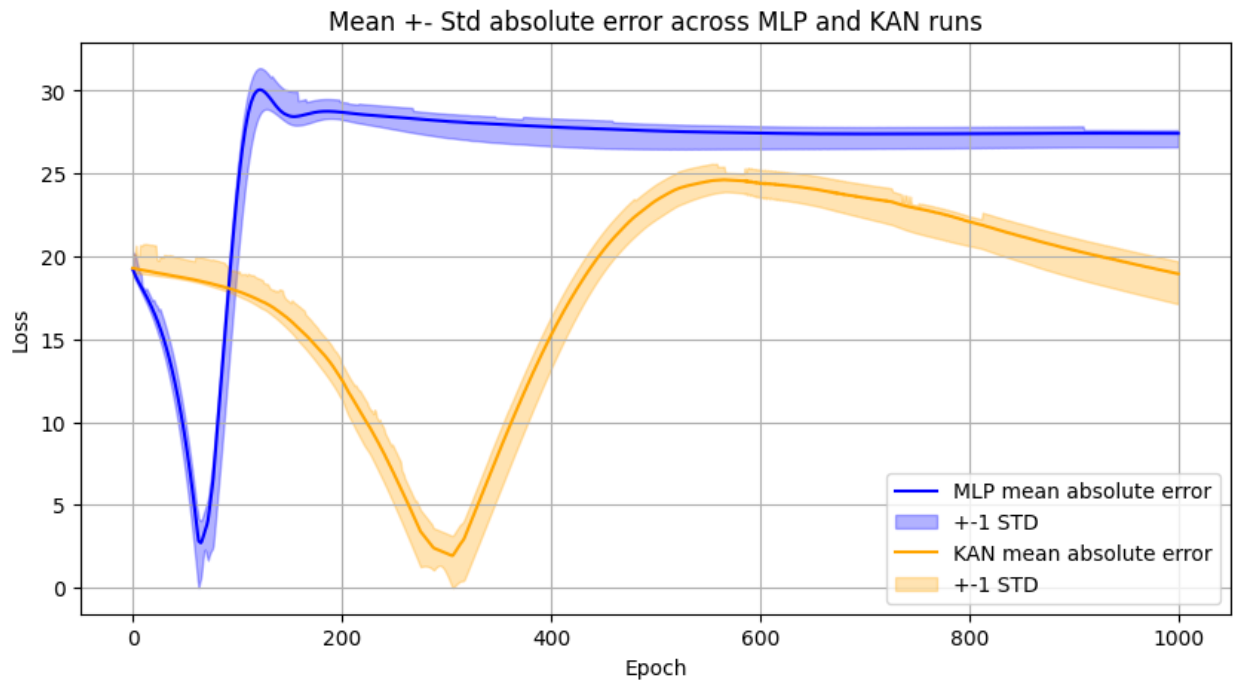


Figure 8-26: Absolute error over 1000 epochs for both models.

The L2 norm tracking reveals that the KAN interestingly was further away, but still was able to reach a better objective value than the MLP. Also, the spread from figure 8-26 indicates again that the KAN explored the search space better. Also, in figure 8-27, both models dip sharply before increasing, then gradually decreasing again, which could indicate that both models found their way out of a local optimum.

## Convergence time and training time

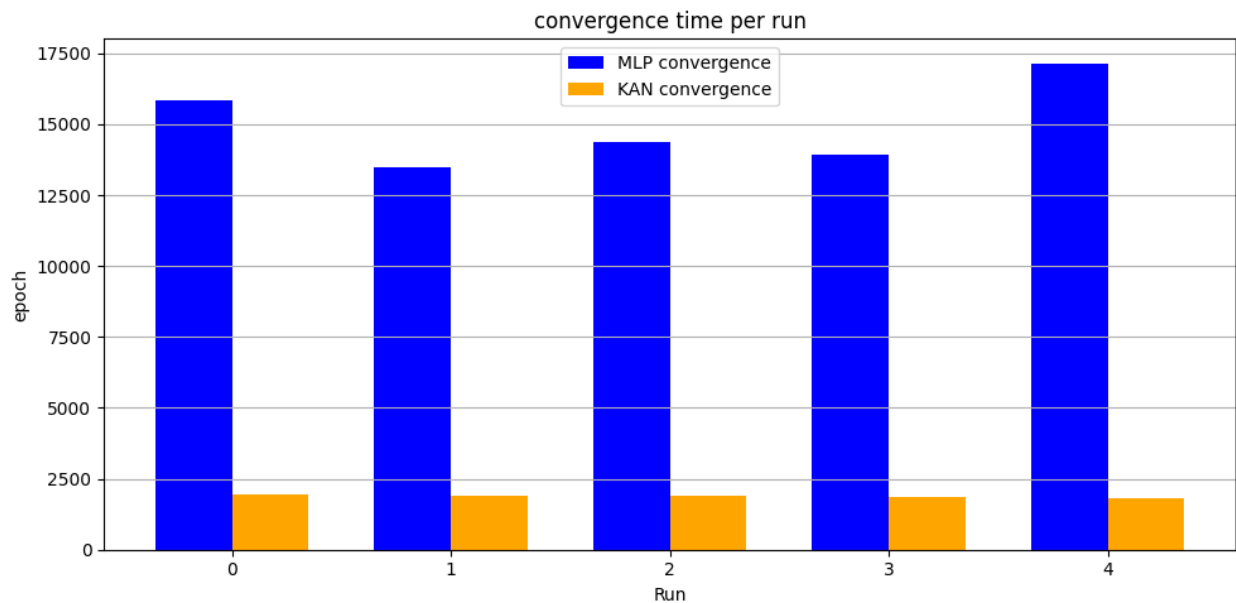


Figure 8-27: Graph of convergence times across all runs for both models.

The MLP was significantly faster to train, requiring only around 61 seconds per run on average. In comparison, the KAN's training time was over 214 seconds per run, with a substantial spread in times, with a standard deviation of 82.3 seconds. This inconsistency suggests that KAN's spline-based architecture may be more sensitive to specific initial conditions, leading to less predictable training behaviour.

Despite its much longer total training time, the MLP required almost 15,000 epochs to stabilise, whereas the KAN converged much earlier, at around 1,896 epochs on average. This again highlights the general pattern seen across problems, where KANs converge much faster due to their architecture.

## Summary

In summary, for this more difficult non-convex optimisation task, the KAN demonstrated better overall performance. It found better objective values more consistently and converged in far less epochs, despite needing more training epochs. The MLP decreased rapidly in loss, however it struggled to match the KAN's solution quality.

## Cross-problem Comparison

In problem 1, the KAN achieved slightly better mean and best objective values, at 1.404 vs 1.413, and converged much faster, requiring only around 750 epochs compared to the MLP's 17000, which is an improvement of over 20 times. Importantly, the KAN was also more consistent, shown by its lower standard deviations across most metrics. While its final loss was marginally higher, the KAN outperformed the MLP on error measures like MAE and RMSE. However, even though it took fewer epochs to converge, the spline operations and grid refinement steps meant that the actual run time was larger.

In problem 2, the performance difference widened further. This problem posed a tougher challenge for both models, yet the KAN managed to find better objective values more consistently, with a mean of 28.18 vs the MLP's 32.52, and with lower variation. Notably, it converged in far fewer epochs, although its total training time increased significantly to 214s compared to the MLP's 61s. The KAN's structure together with the grid refinement approach likely allowed it to better track the local variations in the non-convex landscape.

Overall, the results highlight the KAN's strength in solving non convex optimisation problems. As the dimensionality and complexity of the problems increased, the performance difference between the MLP and KAN was only exacerbated: the KAN is more parameter-efficient, converges faster in terms of iterations, and consistently finds higher quality solutions, even if this requires increased computational time. In contrast, while MLPs are quicker to train, they often struggled to fully exploit the problem landscape, especially in problem 2.



# Stochastic Optimisation

## Problem 1 – Lyapunov Optimisation

### Description

This problem explores a real-time decision-making scenario in the form of queue management under stochastic arrivals and constrained resources, inspired by Lyapunov drift-plus-penalty theory [25]. The formulation represents challenges in wireless communication networks, cloud systems, and real-time energy-aware scheduling, where the aim is to balance queue stability with efficiency under constrained resources. The objective is to dynamically optimise service rates over time to stabilise a queue, without full knowledge of future demand, while respecting energy and delay constraints. In the objective function, the goal is to minimise a trade-off function at each step:

$$\min_{\mu_t} \sum_{t=0}^T (\Delta(Q_t) + V \cdot P(\mu_t)) \quad (9.1)$$

where:

- $\Delta(Q_t)$  is the Lyapunov drift term (change in queue backlog)
- $P(\mu_t)$  is a penalty function, here implicitly linked to energy use
- $V$  is a control parameter balancing stability vs performance

Subject to:

$$Q_{t+1} = \max(Q_t + A_t - \mu_t, 0)$$

$$A_t \sim \text{Poisson}(\lambda), \quad \lambda = 3$$

$$\sum_{t=0}^T \mu_t \leq 400 \quad (\text{Energy budget})$$

$$Q_t \leq 20 \quad \forall t \quad (\text{Delay constraint})$$

$$0 \leq \mu_t \leq 5 \quad \forall t \quad (\text{Service rate limits})$$

Where:

item  $Q_t$ : queue size at time  $t$

item  $A_t$ : number of random arrivals at time  $t$

item  $\mu_t$ : service rate (control variable)

item  $V$ : control parameter balancing stability and cost

This essentially means that each time step, new jobs arrive according to a Poisson process with unknown mean, to simulate volatility.  $\lambda = 3$  allows for a system that is not idle nor overloaded, providing a moderately loaded queue.

The system maintains a queue whose size evolves over time based on arrivals and service, using the constraint  $Q_{t+1} = \max(Q_t + A_t - \mu_t, 0)$ . The new queue size is the current queue size + job arrival rate – service rate.

The optimisation policy is implemented over  $T = 1000$  time steps, with the service rate  $\mu_t$  dynamically adapted using heuristic rules based on queue size and remaining energy. This allows the system to respond to real time stochastic input while attempting to maintain queue stability and meet operational constraints.

- **MLP:**
  - Architecture: Fully-connected network with an input size based on queue size, energy level, and a past window of arrivals, followed by two hidden layers of 32 neurons each with ReLU activations and a single output neuron. A learning rate of 0.001 is used.
  - Loss function: MSE between predicted service rates and true analytically determined service rates.
  - Training strategy: Supervised learning using synthetic data generated via Lyapunov-based optimisation, trained over 1000 epochs with the Adam optimiser. A grid size of 5 is used.
- **KAN:**
  - Architecture: Width configuration [window size + 2, 6, 6, 1] with grid-based spline activations and initial grid resolution 5. A learning rate of 0.001 is used.
  - Loss function: MSE loss, combined with penalties to enforce service rate bounds.
  - Training strategy: Supervised learning using the same dataset as the MLP, with grid refinements applied during training to improve local approximation.

At a glance, the key results reached are as follows:

	Queue size	Service rate	Jump count
<b>MLP</b>	18.96	3.25	63
<b>KAN</b>	17.08	3.03	50
<b>Analytical</b>	18.95	2.85	55

Table 9-1: Summary of final results reached.

## Analysis

Applying neural networks to this problem revealed some key insights into how the KAN behaved in terms of rapidly adjusting to the data.

Metric	MLP	KAN
Mean Queue Size	18.963141	17.458479
Std Queue Size	0.020900	0.892972
Mean Service Rate	3.198799	2.900055
Std Service Rate	0.049452	0.067015
Mean Total Energy Used	400.394250	400.000000
Mean Jump Count	67.000000	61.000000
Mean Final Loss	0.679790	0.468642
Training Time (s)	2.472747	136.107064

Table 9-2: Metrics comparison table

### Queue size and service rate performance

Both models managed to learn reasonable queue control strategies, but the KAN model consistently outperformed the MLP in managing system stability. The KAN achieved a lower mean queue size of 17.46 compared to the MLP's 18.96, suggesting that it was able to regulate the queue more effectively. Also, the KAN maintained a slightly lower average service rate of 2.90 compared to 3.20 for the MLP, with a narrower spread in standard deviation. This indicates that the KAN made steadier, less aggressive service decisions to balance throughput and queue stability.

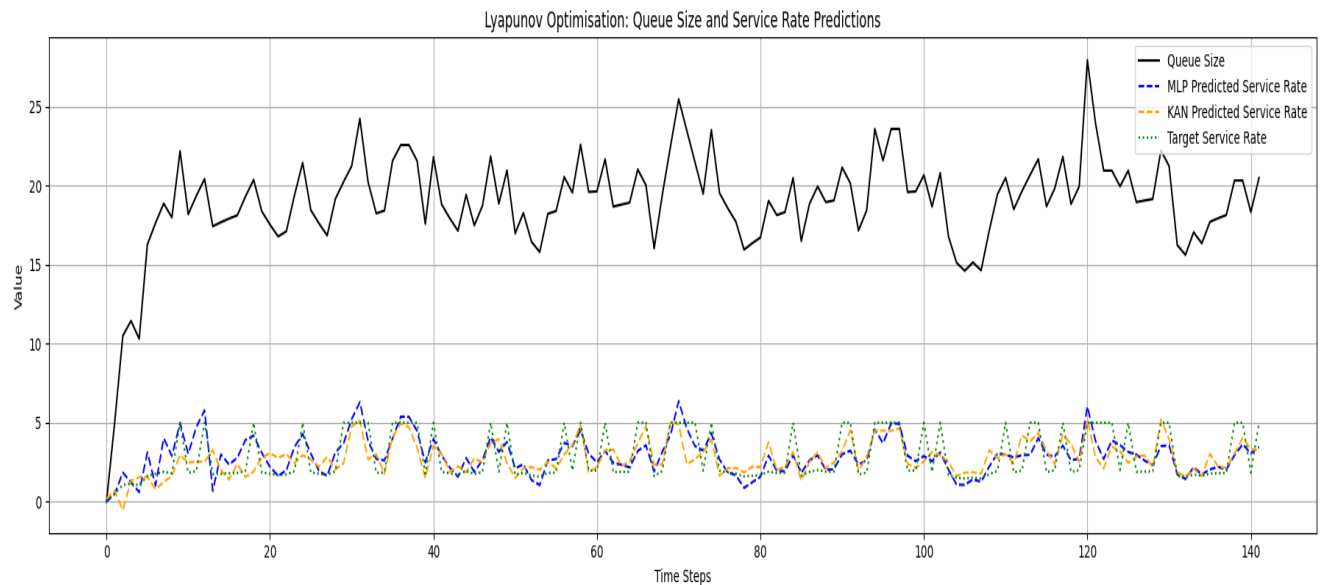


Figure 9-1: The service rate for each model over time. The KAN models the target service rate better than the MLP.

Figure 9-1 shows how the MLP was more variable in service rate compared to the KAN, which was smoother and better able to model the target service rates.

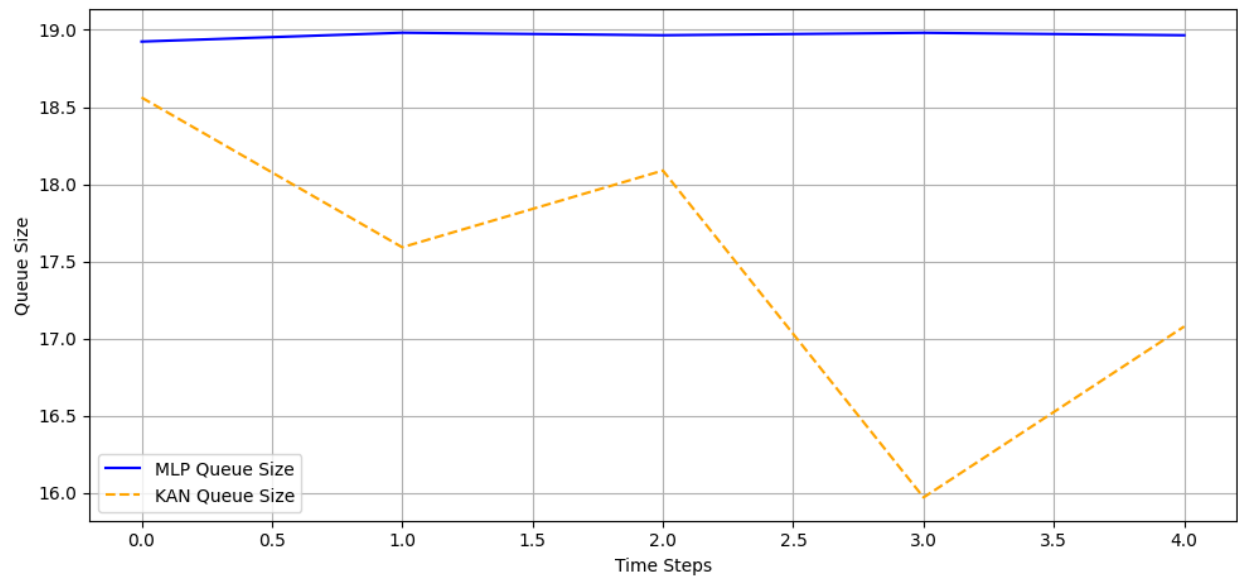


Figure 9-2: The queue sizes across runs. The MLP maintains a near constant queue size whereas the KAN is more variable.

Across the 5 runs, the KAN had a variable average queue size, whereas the MLP was very consistent at around 18.96.

### Training loss behaviour

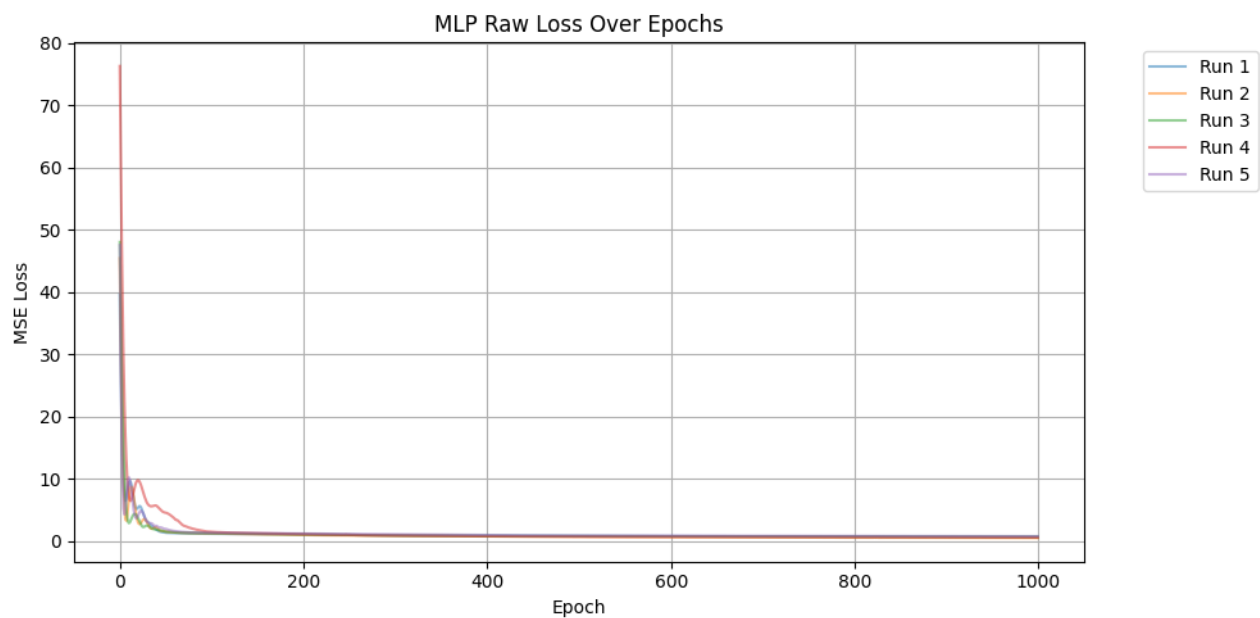


Figure 9-3: MLP loss over 1000 epochs.

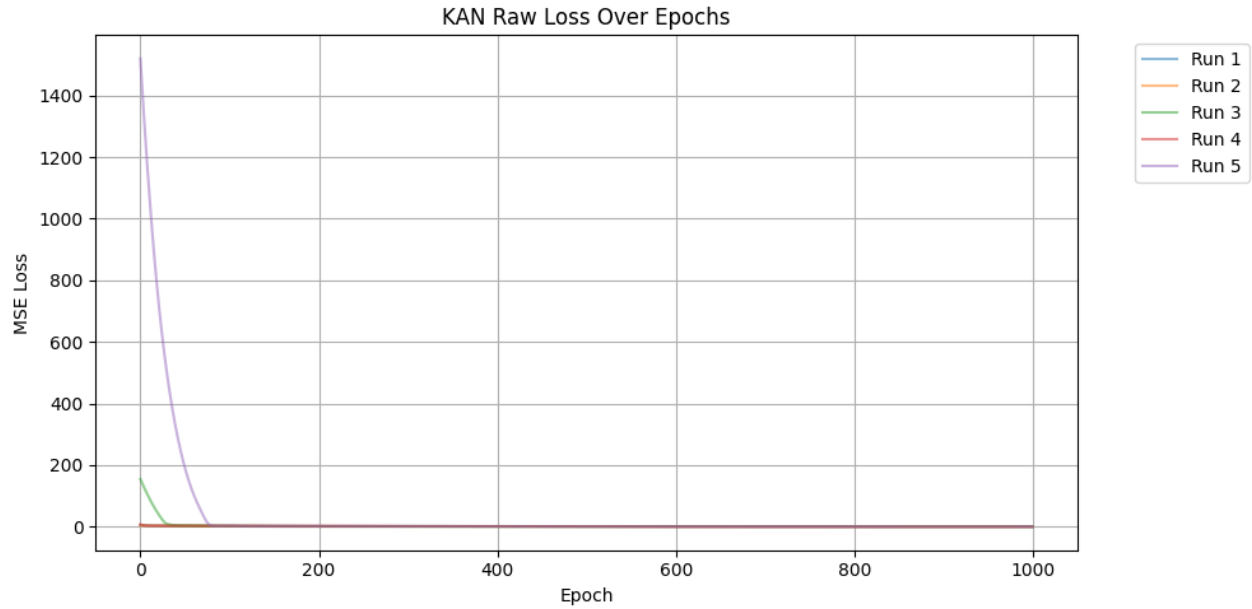


Figure 9-4: KAN loss over 1000 epochs.

In terms of training loss, the KAN again showed its advantages. It reached a lower mean final loss of 0.469, compared to the MLP's 0.680. This lower loss could suggest that the KAN model was better able to map the system state to service actions, while respecting the energy and stability constraints imposed. In figure 9-5, the MLP is shown to start with low loss, but rapidly plateau, whereas the KAN gradually decreases in loss and eventually surpasses the MLP.

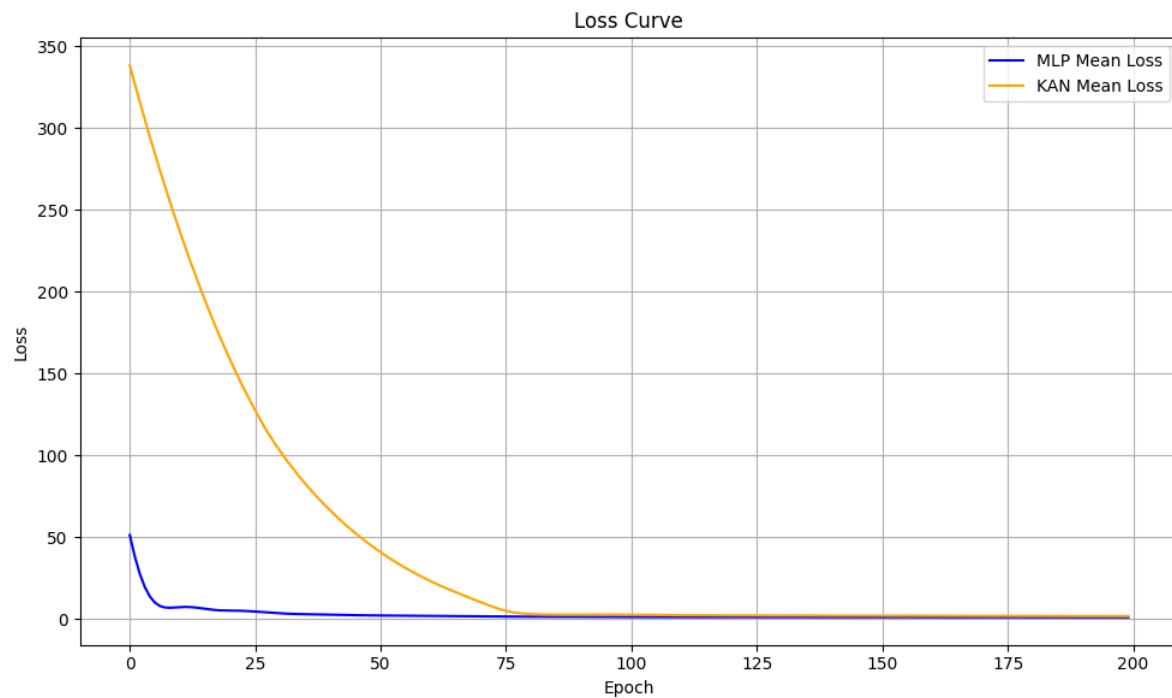


Figure 9-5: Loss curves of both models overlaid, restricted to the first 200 epochs.

## Energy budget and jump count

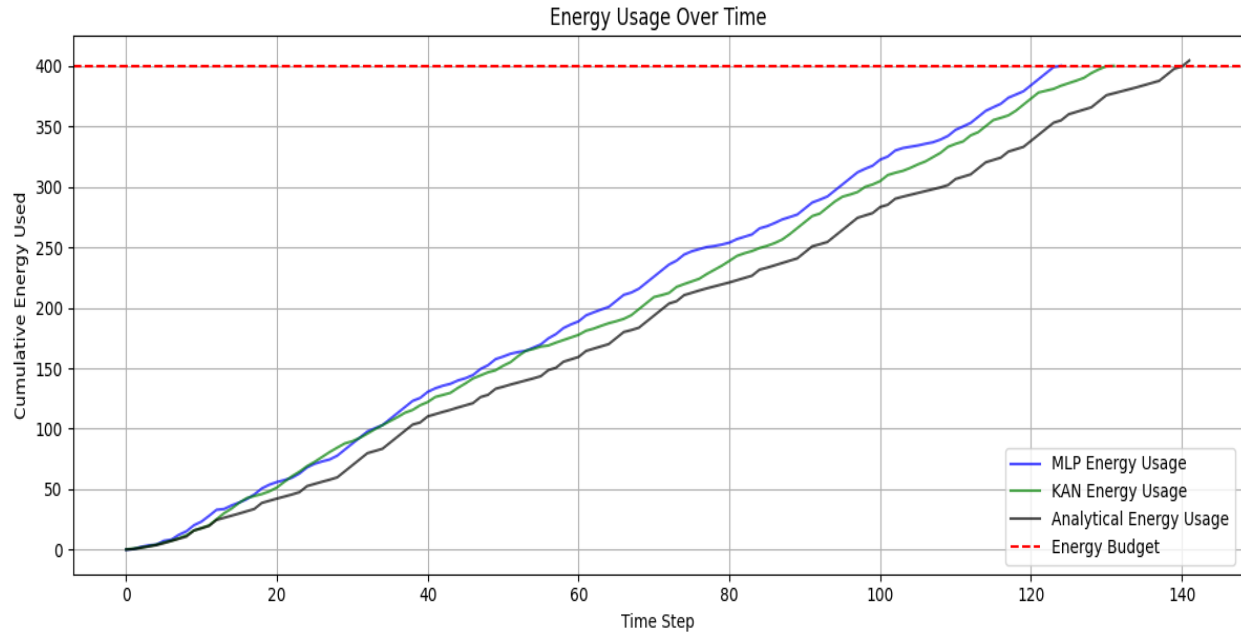


Figure 9-6: Graph of energy usage over time. The KAN uses energy more conservatively than the MLP, and achieves better results because of it.

Both models respected the strict energy budget constraint of 400 energy units. However, the KAN was able to manage it better than the MLP, and ended up closer to the analytical model's performance.

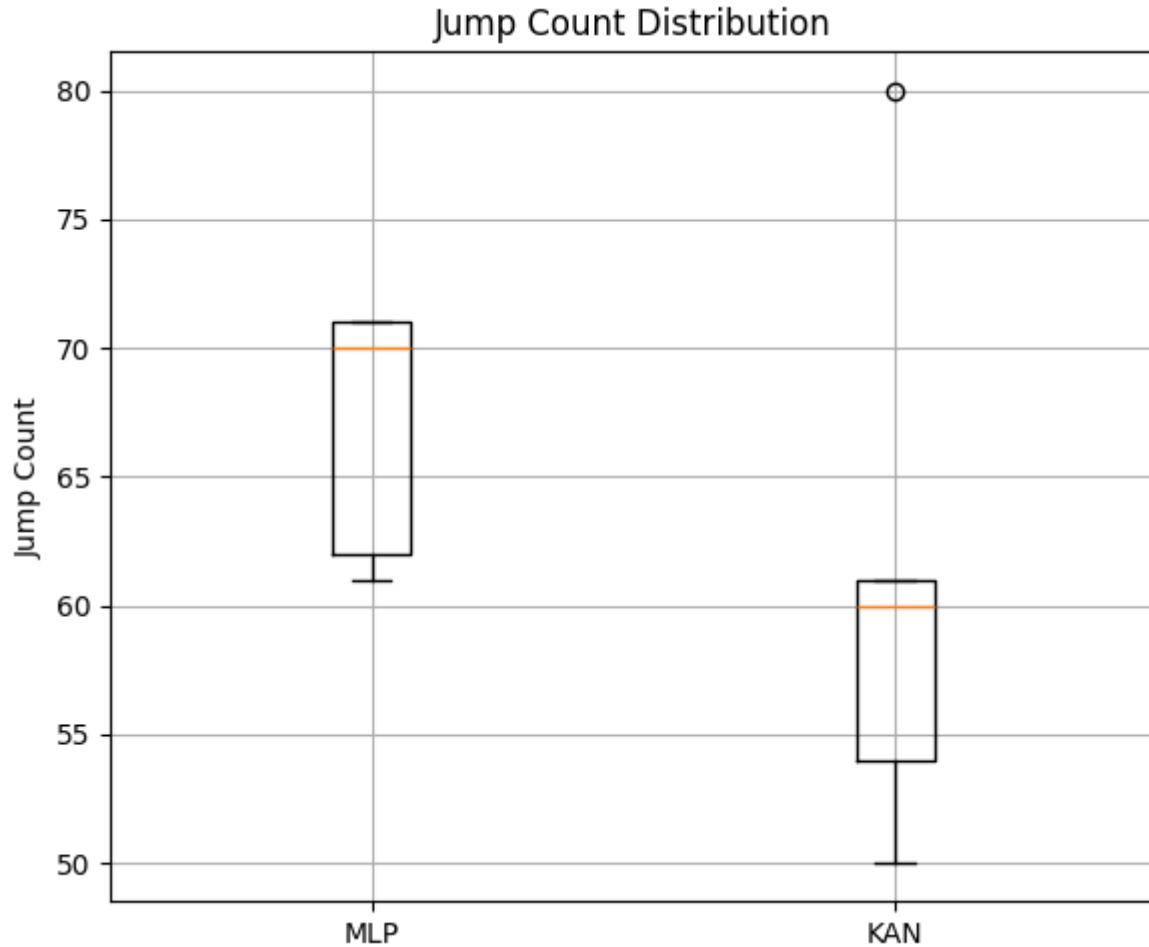


Figure 9-7: Box plot representing jump counts of both models. The KAN had a lower jump count, indicating that it was more stable.

In terms of policy smoothness, measured via jump count/abrupt service rate changes, the KAN again outperformed the MLP, recording only 61 jumps on average versus the MLP's 67. This shows that the KAN made more consistent, gradual adjustments, which could be especially important in real world scenarios where inconsistent shifts can be impractical.

### Training time

Training time was a major difference between the two models. The MLP was very fast, taking only 2.47 seconds, whereas the KAN required significantly longer, at approximately 136.1 seconds per run. This difference is not surprising, given the KAN's more complex internal structure, but it may be a significant practical consideration depending on the use case.

### Summary

Overall, although the MLP had significantly faster training, the KAN consistently achieved superior performance in terms of system regulation and policy smoothness. Its stronger ability to minimise queue size and maintain a smooth service policy while adhering to strict energy

budgets and constraints makes it particularly attractive for applications where stability and robustness under uncertainty are critical.

## Problem 2 – Budget Constrained Price Optimisation

### Description

Problem 2 simulates a stochastic price optimisation scenario from a seller's perspective, where the goal is to maximise revenue from multiple products sold to budget-constrained customers. The challenge lies in setting prices that consider how demand varies with price and how each customer's limited budget constrains their purchase behaviour:

$$\max_p \left( \sum_{i=1}^n p_i \cdot D_i(p_i) \right) \quad (9.2)$$

Subject to:

$$\sum_{i=1}^n \left( p_i \cdot \frac{D_i(p_i)}{\sum_j D_j(p_j)} \cdot d_i \right) \leq B_k, \quad \forall k = 1, \dots, m$$

$$0.01 \leq p_i \leq 10.0, \quad \forall i = 1, \dots, n$$

where:

- $D_i(p_i) = d_i \cdot \exp(-\alpha_i p_i)$ : the demand for product  $i$ , modelled as exponentially decaying with respect to its price
- $d_i$ : base demand for product  $i$
- $\alpha_i$ : price sensitivity for product  $i$
- $p_i$ : price of product  $i$
- $B_k$ : budget of customer  $k$
- $\frac{D_i(p_i)}{\sum_j D_j(p_j)} \cdot d_i$ : expected share of product  $i$  purchased by customer  $k$

Essentially, the problem is to **set the prices of a number of products** such that the **revenue is maximised without the customers budgets being exceeded**, while taking into account the demand for the product, which changes with pricing.

This problem was chosen for its practical relevance in dynamic pricing and non-convex constraints, making it a suitable candidate to evaluate the adaptability of neural approaches like MLPs and KANs.

Two models were trained to solve this optimisation problem:



- **MLP:**
  - Architecture: Fully-connected network with an input of products and two hidden layers of 64 neurons each (ReLU activations), outputting a single predicted revenue value. A learning rate of 0.001 is used.
  - Loss function: MSE between the predicted and actual revenue over a synthetically generated training set.
  - Training strategy: Supervised learning using 5000 epochs of random feasible product prices sampled in the range [1, 10], trained with the Adam optimiser. There are 10000 samples selected.
- **KAN:**
  - Architecture: Width configuration [number of products, 6, 6, 1] with spline-based activations, using a grid resolution of 10 for initial training. A learning rate of 0.01 is used.
  - Loss function: MSE loss over predicted revenues.
  - Training strategy: Supervised learning on the same training dataset as the MLP, with iterative updates to spline parameters to fit the pricing function

At a glance, the key results reached are as follows:

	<b>Optimal prices</b>	<b>Final revenue</b>
<b>MLP</b>	9.0877, 1.0605, 2.5014, 1.4073, 1.9381	516.52
<b>KAN</b>	8.6761, 1.1313, 2.1294, 2.7644, 1.2074	536.42
<b>Analytical</b>	7.7952, 1.4864, 2.6115, 1.7930, 1.0907	548.85

Table 9-3: Summary of final results reached. The KAN achieves a higher revenue than the MLP.

## Analysis

Again, the KAN showed greater ability in consistently reaching better results.

<b>Metric</b>	<b>MLP</b>	<b>KAN</b>
Mean Final Revenue	508.757324	525.401794
Std Final Revenue	17.577276	7.989663
Mean Computation Time (s)	70.376559	371.361592
Std Computation Time (s)	6.59581	19.342401
Mean Training Loss	2877.080853	5819.957858
Std Training Loss	121.190358	662.542752

Table 9-4: Metrics comparison table

## Revenue performance

Both models were able to learn effective pricing strategies under budget constraints, but the KAN clearly outperformed the MLP. The KAN model achieved a higher mean final revenue of 525.40, compared to the MLP's 508.76. Also, the KAN exhibited much lower variability between runs, with a standard deviation of only 7.99, whereas the MLP's standard deviation was

significantly higher at 17.58. This shows that the KAN was not only able to achieve better results but did so more consistently, demonstrating greater robustness across different training runs.

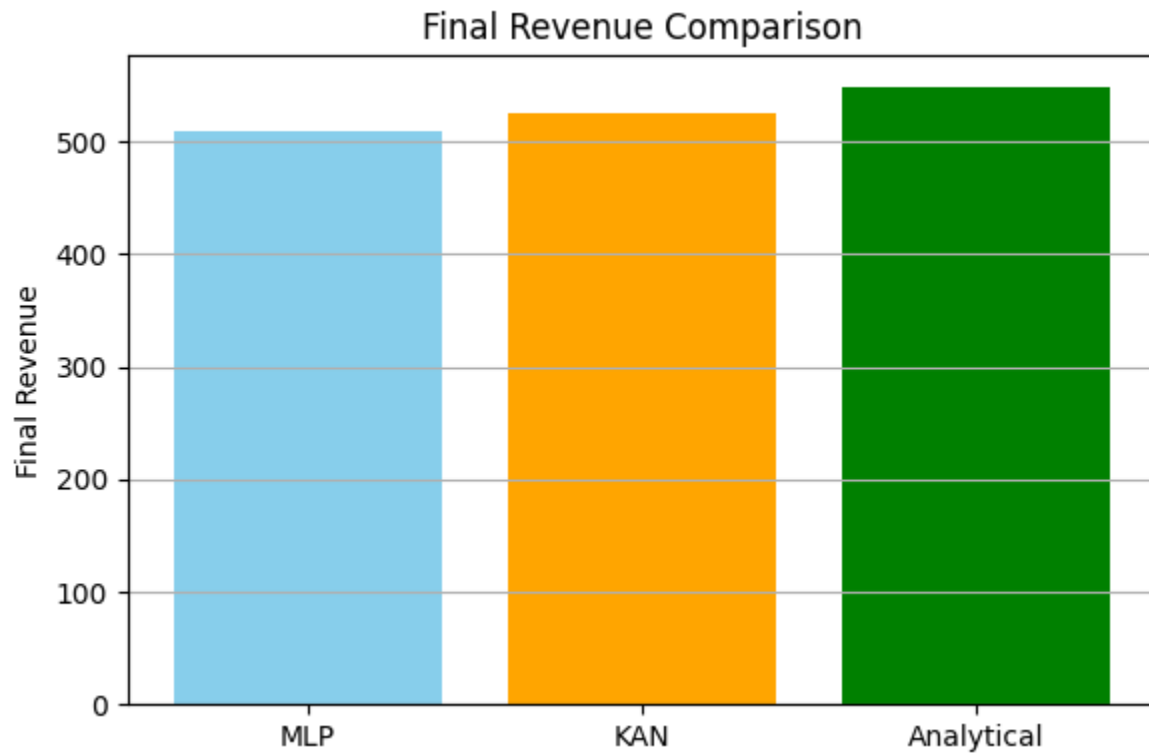


Figure 9-8: Visualisation of final revenues reached by the models. The KAN reaches a better mean revenue than the MLP.

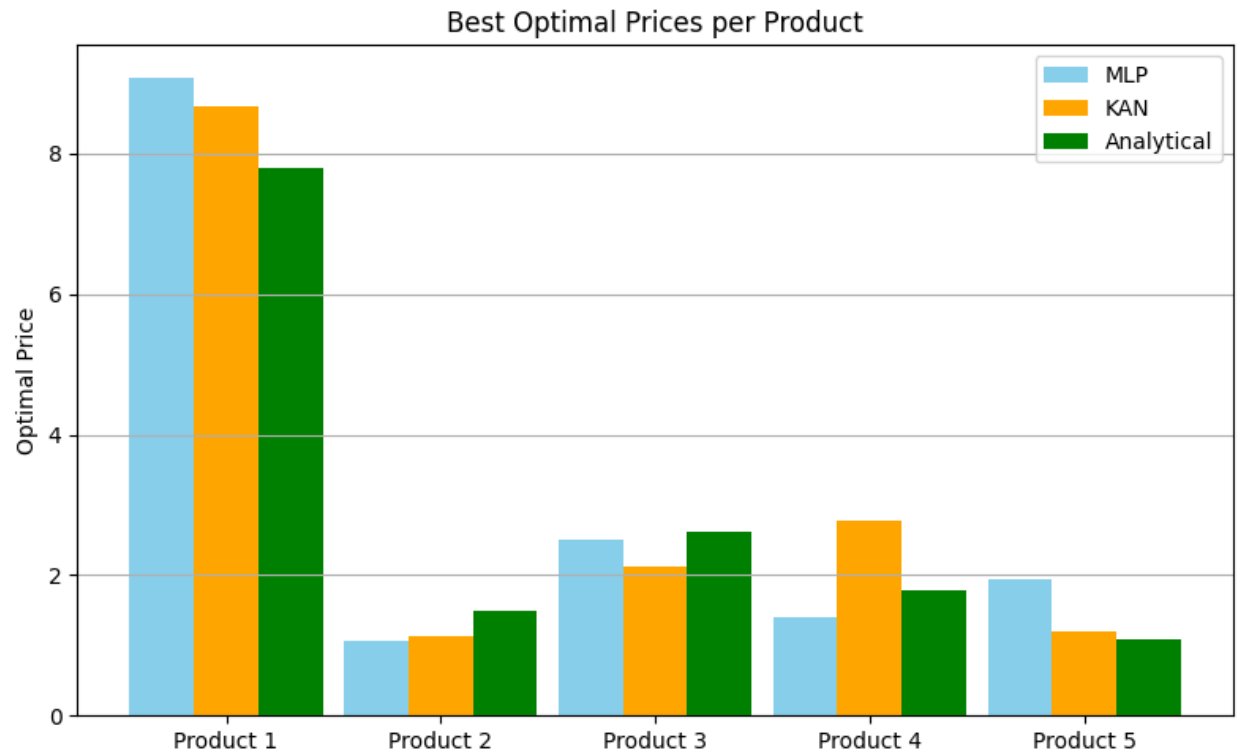


Figure 9-9: Visualisation of the optimal prices reached. The KAN prices the items closer to the analytical solution than the MLP does.

In figure 9-9, the distribution of product prices is shown. The KAN is able to price its items better than the MLP can, given the same constraints, and hence reaches a better revenue.

### Loss behaviour

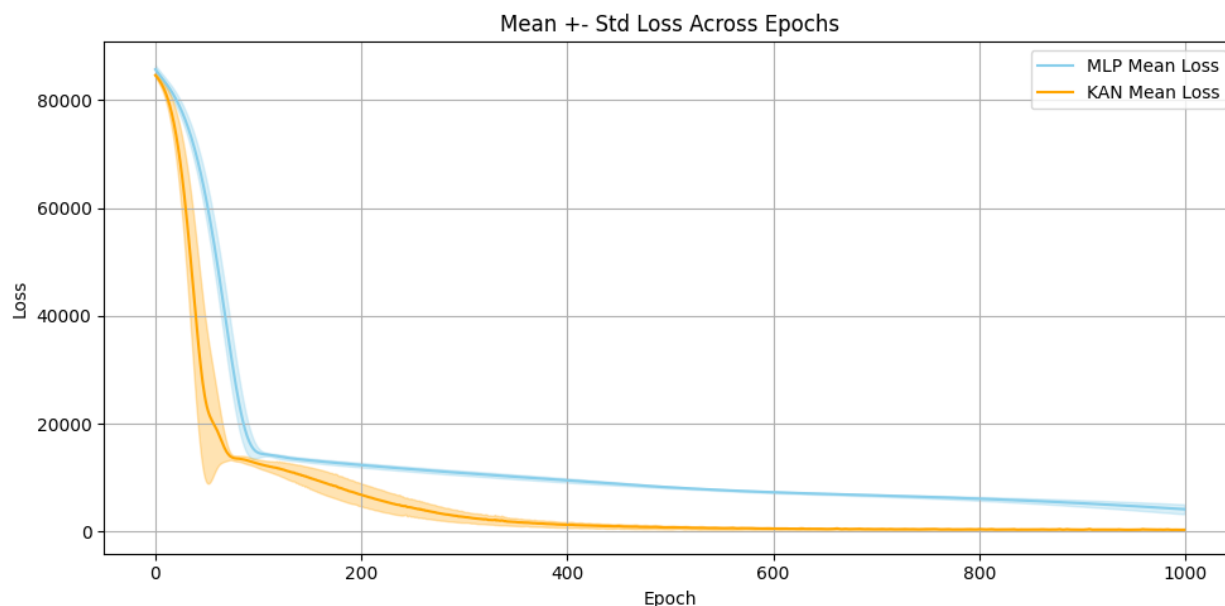


Figure 9-10: Loss curves of both models overlayed. The KAN reaches a lower loss faster.

Interestingly, despite achieving better revenue outcomes, the KAN ended with a substantially higher mean final training loss of 5819.96, compared to the MLP with 2877.08. This highlights the important observation that lower loss does not automatically translate to better real-world performance. While the MLP achieved a better fit to the training targets in terms of MSE, the KAN was more successful in maximising the actual revenue, which was the true goal of the task.

### Model stability and training time

The KAN also demonstrated greater stability across runs, not only in terms of output revenue but also in computational time, showing a much lower standard deviation in both metrics compared to the MLP. This suggests that the KAN's training dynamics were more reliable, whereas the MLP showed higher variability depending on initialisation and training conditions.

However, in terms of computational cost, the MLP was significantly faster, completing training in around 70.4 seconds on average, compared to the KAN's 371.4 seconds. This longer training time for the KAN reflects the overhead associated with its more complex architecture and grid refinement mechanisms. Even so, considering the substantial revenue gains achieved by the KAN, in situations where the final solution quality is critical, this disadvantage could be justified.

### Summary

In this stochastic pricing task, the KAN model was shown to outperform the MLP across most important metrics, achieving higher revenue, greater consistency, and more stable behaviour at the cost of significantly longer training times. It's strength in modelling non-convex relationships with constraints likely made it especially well-suited to this kind of optimisation challenge.

In contrast, the MLP, while faster and achieving lower training loss, did not match the KAN's real-world performance.

## Cross-problem Comparison

When comparing the two stochastic optimisation problems, some clear trends emerge in terms of how the MLP and KAN models handle uncertainty and dynamic constraints. In problem 1, both models managed to learn reasonable policies, but the KAN demonstrated a clear advantage in policy smoothness and queue stability. The KAN achieved a lower mean queue size and fewer large service-rate jumps, indicating that it was better at balancing service rates and energy consumption dynamically. However, this came at a significant computational cost, in that the KAN took substantially longer to train than the MLP, highlighting a trade-off between solution quality and computational efficiency.

In contrast, for problem 2, the KAN maintained its advantage in final performance but still underperformed in computational time relative to the MLP. The KAN achieved a higher mean final revenue with lower standard deviation, suggesting that it generalised better across different budget distributions and customer behaviours. While training still took longer for the KAN, the margin was narrower compared to the first stochastic problem, and the gains in solution quality were more pronounced.

Overall, these results suggest that KANs are highly effective at modelling complex stochastic behaviours, particularly when solution stability and robustness under uncertainty are critical. However, their higher computational demands mean that they are more appropriate in scenarios where solution quality is prioritised over strict runtime constraints.

## Discussion

While KANs demonstrated superior convergence and stability in most cases, their performance trade-offs and the unique challenges of discrete optimisation revealed important nuances.

In the linear optimisation problems, both the MLP and KAN were able to learn the optimal solutions with high precision, but the KAN stood out in several aspects. KANs consistently converged in fewer epochs, in under 1000 compared to over 10,000 for the MLP, despite having far fewer parameters. This suggests that the B-spline activations in KANs allow for more efficient representation of linear regions in the objective space. Even though MLPs had more parameters, they didn't necessarily translate to better performance, and in fact, were more prone to noisy convergence early in training. KANs, by contrast, improved steadily and achieved slightly higher objective values overall, making them more appealing in situations where convergence speed and compact architecture are important.

The integer problems revealed more nuance. Both models were designed to handle these problems using a discrete search space, training only on integer-valued inputs. Interestingly, the MLP model converged very quickly, possibly due to the fact that the search space was restricted heavily, whereas the KAN model had a far higher convergence time - there was an order of magnitude in difference in epochs. On simpler problems, the MLP outperformed the KAN in terms of both convergence and final solution quality. However, in more complex formulations, the KAN was able to produce comparable or superior objective values, using fewer parameters. This suggests that, despite the MLP performing better overall, KANs may scale better with problem complexity, even though they are not specifically designed for discrete optimisation. This could be a potential area for future improvements, possibly through hybrid approaches or integration with symbolic solvers.

KANs consistently demonstrated better resilience to local minima and showed more stable convergence across runs. For example, in Non-Convex Problem 1, the MLP achieved marginally better objective values, but required 20 times more epochs and was highly sensitive to initial conditions. KANs stabilised quickly and maintained tighter variance in results. In the more complex Problem 2, both models struggled, but evidently the KAN achieved objective values closer to the optimal, whereas the MLP showed a tendency to get stuck in local optima. Despite a higher model size, the KAN's convergence behaviour and ability to model the landscape with fewer fluctuations indicated a clear advantage in robustness.

Stochastic optimisation highlighted the trade-off between quality and efficiency most clearly. In problem 1, the Lyapunov queue management task, the KAN achieved lower average queue sizes, smoother service policies with lower jump counts, and more consistent adherence to the energy budget. Overall, the KAN had more accurate service rate predictions, compared to the MLP which was not able to model complex mappings between queue state and service decisions nearly as well. However, in terms of computational cost, the MLP took only 3 seconds,

compared to the KAN, which consistently took over 130 seconds. In real-time systems or low-resource environments, this extra cost might be prohibitive. In the budget constrained price optimisation task, the KAN again achieved better mean revenue with lower variability across runs, even though their training loss was significantly higher. This reinforces that in real-world tasks with constraints and noise, KANs are able to achieve reliable, feasible outcomes better.

What is particularly notable is that KANs showed strong performance across every category, without needing specialised architectures or hyperparameters for different problems. It suggests that KANs could be used as a flexible and interpretable tool for solving optimisation problems where traditional solvers struggle, especially those involving non-linear dynamics, complex constraints, or noisy data. The ability to refine the grid mid-training is also a powerful mechanism for balancing global exploration with local fine-tuning. While they did not always outperform MLPs on raw loss metrics or compute time, they consistently demonstrated more stable training, stronger convergence properties, and higher quality solutions in complex settings.

Even so, KANs still have limitations. Their training times are slower, they require more careful tuning of grid sizes and refinement steps, and their current implementations lack the same infrastructure as MLPs. Nevertheless, their strengths suggest that KANs are well-suited to optimisation tasks that require interpretable, stable, and constraint-aware learning.

## Conclusion

This project set out to evaluate the effectiveness of KANs in solving a broad range of mathematical optimisation problems. Specifically, it investigated how KANs perform compared to standard MLPs across four key categories: linear, integer, non-convex, and stochastic optimisation. The aim was to assess whether the structural innovations of KANs translate into tangible benefits when tackling a variety of optimisation tasks.

To ensure a robust and fair evaluation, each problem type was represented by two distinct problems, one simpler and one more complex. All problems were also solved analytically using traditional optimisation tools, providing ground-truth baselines for comparison. MLPs and KANs were then trained using supervised or policy-learning strategies, with multiple metrics being used to assess performance.

The results demonstrated that KANs are able to outperform MLPs across most tasks, especially in problems involving non-linear objectives, constraints, or stochastic behaviour. KANs required significantly fewer training epochs to converge, produced more stable solutions across initialisations, and exhibited a better ability to respect constraints without excessive regularisation. This was most evident in the non-convex and stochastic settings, where MLPs struggled with local minima or oscillating training curves, while KANs learned smoother, more reliable objective surfaces.

Another important aspect of KANs was that they were found to be very generalisable. The same model architecture was able to handle problems of varying structure and domain without task-specific tuning. This suggests that KANs may serve as a strong foundation for general-purpose optimisation in machine learning, particularly when classical solvers are infeasible or when data-driven approximation is preferred.

Despite these strengths, several limitations remain. KANs are more computationally expensive per epoch, and their performance in very high-dimensional problems has yet to be fully assessed. Furthermore, the need to manage spline grids and regularisation parameters introduces some complexity not present in simpler MLP models. Nevertheless, the advantages they provide in terms of convergence speed, interpretability, and constraint adherence may justify these drawbacks, especially in constrained or hybrid optimisation contexts.

In summary, this report provides strong empirical evidence that KANs are a valuable tool in the optimisation landscape. They offer a new way to approach real-world problems, by bridging the gap between data-driven modelling and constrained mathematical reasoning. In practical terms, this opens up applications in pricing, logistics, scheduling, and network control, where solution spaces are often complex and neural networks must make real-time or approximate decisions.



# Bibliography

- [1] Z. Liu, "KAN: Kolmogorov–Arnold Networks," 2024. [Online]. Available: <https://arxiv.org/abs/2404.19756>.
- [2] S. K. S. D. Benjamin Koenig, "KAN-ODEs: Kolmogorov–Arnold network ordinary differential equations for learning dynamical systems and hidden physics," 2024.
- [3] G. Cybenko, "Approximation by superpositions of a sigmoidal function," 1989.
- [4] L. Alzubaidi, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," 2021. [Online]. Available: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8>.
- [5] A. Kolmogorov, "On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables.," 1956.
- [6] S. Academy, "Kolmogorov-Arnold Networks (KANs) - What are they and how do they work?," YouTube, 2024. [Online].
- [7] S. Somvanshi, S. A. Javed, M. M. Islam, D. Pandit and p. Subasish Das, "A Survey on Kolmogorov-Arnold Network," 2024. [Online]. Available: <https://arxiv.org/pdf/2411.06078>.
- [8] Y. B. A. C. Ian Goodfellow, Deep learning, 2017.
- [9] Z. LIPTON, The Mythos of Model Interpretability, 2018.
- [10] L. V. Stephen P. Boyd, Convex Optimization, 2004.
- [11] G. N. LA Wolsey, Integer and combinatorial optimization, 1999.
- [12] A. Shapiro, Lectures on Stochastic Programming, Mathematical Programming Society Philadelphia, 2009.
- [13] V. Klee and G. Minty, "How Good Is the Simplex Algorithm?," 1972. [Online]. Available: <https://www.scirp.org/reference/referencespapers?referenceid=1869486>.
- [14] N. Karmarkar, "A new polynomial-time algorithm for linear programming," 1984. [Online]. Available: <https://www.scirp.org/reference/referencespapers?referenceid=1869486>.

- [15] D. Wu and A. Lisser, "A deep learning approach for solving linear programming problems," 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0925231222014412>.
- [16] A. D. AH Land, An automatic method for solving discrete programming problems, 2009.
- [17] D. Yilmaz, "Learning Optimal Solutions via an LSTM-Optimization Framework," 2022. [Online]. Available: <https://arxiv.org/pdf/2207.02937>.
- [18] S. W. J Nocedal, Numerical optimization, 1999.
- [19] P. K. P Jain, Non-convex optimization for machine learning, 2017.
- [20] Encord, "Time Series Predictions with RNNs," 2023. [Online]. Available: <https://encord.com/blog/time-series-predictions-with-recurrent-neural-networks/>.
- [21] D. Kingma and M. Welling, Auto-Encoding Variational Bayes, <https://arxiv.org/abs/1312.6114>, 2022.
- [22] Z. Liu, "Pykan," [Online]. Available: <https://github.com/KindXiaoming/pykan>.
- [23] F. H. Ilya Loshchilov, "Decoupled Weight Decay Regularization," 2017.
- [24] S. Community, "scipy.optimize.linprog," [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html>.
- [25] L. Bracciale and P. Loreti, "Lyapunov Drift-Plus-Penalty Optimization for Queues With Finite Capacity," 2020.

## Appendices

All code and notebooks, including additional tables and figures, are available at the following GitHub repo: <https://github.com/arjy143/Neural-Networks-for-Optimisation/>