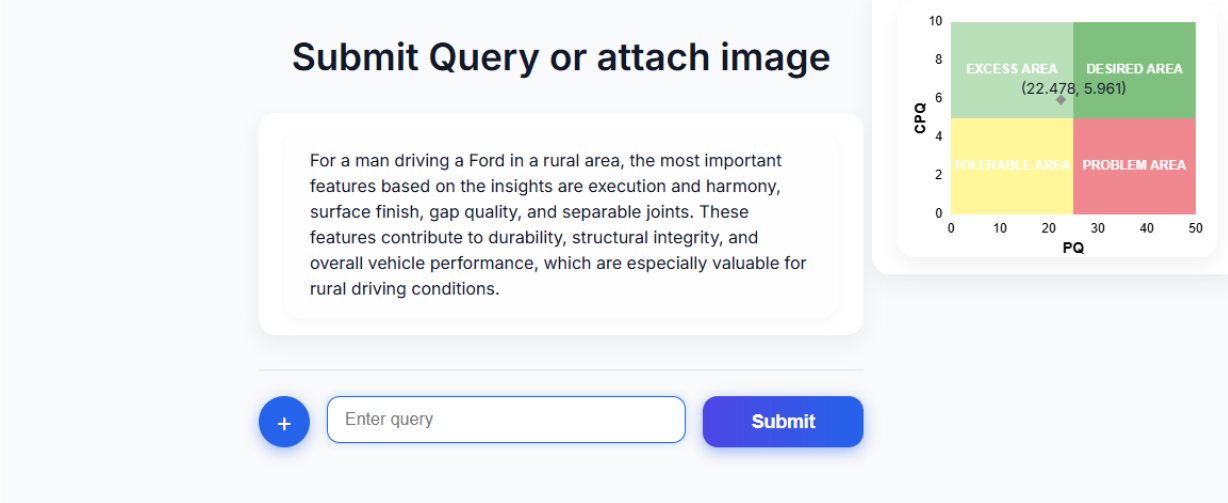
CHALMERS
UNIVERSITY OF TECHNOLOGY




# NextGen AI Agent for Perceived Quality of Cars

Master's thesis in Embedded Electronic System Design

Erik Eliasson
Emil Johansson

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

# NextGen AI Agent for Perceived Quality of Cars

Erik Eliasson and Emil Johansson


Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2025

NextGen AI Agent for Perceived Quality of Cars
Erik Eliasson, and Emil Johansson



Supervisor: Lars Svensson, Department of Microtechnology and Nanoscience
Company advisor: Kostas Styldis, Intended Future
Examiner: Per Larsson-Edefors, Department of Microtechnology and Nanoscience

Master's Thesis 2025
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: User interface of the final product.

Typeset in LaTeX
Gothenburg, Sweden 2025

NextGen AI Agent for Perceived Quality of Cars
Erik Eliasson
Emil Johansson

Department of Microtechnology and Nanoscience
Chalmers University of Technology

# Abstract

This thesis details an AI-powered tool intended to assist designers in evaluating Percived Quality in the early stages of the design of the development of vechicles. The tool integrates machine learning models, image analysis, and a Retrieval-Augmented Generation system to provide real-time, context-aware insights based on customer data and design attributes. Enabling proactive decision-making aims to reduce costly design iterations and align products more closely with user expectations.

Keywords: AI, perceived quality, RAG, machine learning, image analysis, automotive design, user feedback

# Acknowledgements

We would like to thank our supervisor Lars Svensson and our examiner Per Larsson-Edefors at the for their guidance and support throughout the project. We also want to thank Kostas Styldis at Intended Future for allowing us to do our master thesis at his company and for his valuable input throughout our project.

Erik Eliasson and Emil Johansson, Gothenburg, June 2025

# Contents

# Nomenclature

### all-MiniLM-L6-v2

**Definition:** A lightweight sentence-transformer model developed by SentenceTransformers, designed for generating dense vector embeddings of text for tasks like semantic search, clustering, and similarity comparison. It balances speed and performance, making it ideal for real-time or large-scale applications.

### API (Application Programming Interface)

**Definition:** A set of protocols that allow interaction between different software components.

### Approximate Nearest Neighbor (ANN) Search

**Definition:** An algorithm that retrieves similar vectors efficiently with a trade-off between speed and accuracy.

### Caching

**Definition:** Storing frequently accessed data for faster retrieval.

### Craftsmanship score (CPQ)

**Definition:** Company impression depending on a specific error, for example Flush, Gap or See Thrue Parts.

### Embedding Queries

**Definition:** The process of querying a vector database using embedding representations.

### Embeddings

**Definition:** High-dimensional vector representations of objects such as text or numerical data for efficient similarity search.

### Final Integration

**Definition:** The process of combining all system components into a functional end-to-end solution.

### Hardware Acceleration

**Definition:** The use of GPUs or TPUs to speed up computational tasks like vector searches.

### Indexing Strategies

**Definition:** Techniques used to organize vector data for efficient retrieval.
**Methods:** Approximate nearest neighbor search, quantization.

### Latency Stress Test

**Definition:** Measuring system response times under varying load conditions.

### Large Language Models (LLM)

**Definition:** AI models trained on vast text data to generate human-like responses.
**Examples:** ChatGPT, Claude.

### Parallel Processing

**Definition:** Simultaneous execution of multiple tasks to improve efficiency.

### Perceived Quality (PQ)

**Definition:** Customer impression depending on a specific error, for example Flush, Gap or See Thrue Parts.

### Perceived Quality Framework (PQF)

**Definition:** A model used to assess perceived quality in automotive systems based on subjective dimensions.
**Key Dimensions:** Sensory Perceptions, Craftsmanship, Design Coherence.

### Query Prefetching

**Definition:** A technique that pre-loads queries to improve response times.

### Retrieval-Augmented Generation (RAG) Systems

**Definition:** A system combining vector database retrieval with large language models for intelligent response generation.

### Scalability

**Definition:** The ability of a system to handle increased data or user requests without performance degradation.

### Stress Testing

**Definition:** Evaluating system performance under extreme load conditions.

### Technical Documentation

**Definition:** A detailed guide for developers and engineers to understand system architecture and functionality.

### User Interaction Cases

**Definition:** Simulating real-world user inputs to validate system accuracy and performance.

### User Manual

**Definition:** A non-technical guide for end-users explaining system features and interactions.

### Vector Databases

**Definition:** Databases optimized for storing and retrieving high-dimensional vectors.
**Examples:** Pinecone, Weaviate, FAISS.

### Vectorization

**Definition:** The process of converting PQF elements into numerical vector representations for machine learning and retrieval.

# 1
# Introduction

In the automotive industry, the Perceived Quality (PQ) of a vehicle is important and hard to determine from the beginning. Different features, such as the sound when closing a door, display responsiveness, illumination uniformity, and haptic feedback, influence how the user perceives the quality of the car. These features, and many more, are connected to the embedded system and design of cars. Today, when designing cars, it is very cost-ineffective to start a design and, after a while, restart the design process since it shows that the customer wants something totally different.

Integrating the PQ framework into the design process of a car will reduce these costs while also helping the designer to actually meet and, as long as it is possible, exceed the customers' expectations. Today, integrating this framework remains a challenge since many datasets rely on isolated analysis, limiting their ability to provide actionable, system-level insights [1]. The PQ dataset consists of multidimensional complex data and requires advanced methods and tools to investigate and find relationships between features and their measured PQ to provide engineers and designers with real-time support during the design process of embedded systems.

This thesis project aims to address these challenges by developing a Retrieval Augmented Generation (RAG) system that connects the Perceived Quality Framework (PQF) with state-of-the-art Natural Language Processing (NLP) and embedding technologies. A domain-specific vector database will be created to encapsulate PQF elements such as Sensory Perception (e.g., audio, visual, and tactile signals) and Design Coherence in the context of hardware performance and embedded system behavior. This database will interface with a stable Large Language Model (LLM) to provide engineers with real-time context-aware PQ insights via API-driven applications.

The research focus will be on the potential to improve and advance PQ system integration in the development process of electrical systems and hardware in the automotive industry. The system can be used to allow engineers to make decisions that will positively affect customer satisfaction by fine-tuning certain parts that customers consider important right from the start of the process. This may lead to a reduction in the number of design cycles, and the products will hopefully undergo fewer cycles, which is not only positive for the company but also from an environmental point of view, while meet satisfaction of the customers in an efficient manner.

The RAG system will be controlled from the back-end application developed in this thesis project, which will produce prompts based on the user's question to the system, stored PQ insights, and advanced methods for predicting what customers and craftsmanship consider important. The RAG system will, to some extent, be tied to certain platforms, which also require subscriptions and further development.

## 1.1 Related Work

With the recent technological advancements in the automotive industry, focus has largely shifted from the technical qualities of the product to the PQ of the customer [2]. As most automotive companies continue to refine and perfect qualities like safety features and performance, more of the weight lies in factors like ergonomics, sensory experience, and aesthetic appeal. The comfort of using the product and how it sounds and feels has grown in importance in the last couple of years, and it's essential for automotive companies to not only prioritize functionality but also start focusing on the perception and experience of the customer.

Currently, the PQ of the customers has been gathered by the companies through customer surveys, testing by experts, and material analysis [2]. While these methods do provide useful insight into how the customer perceives the product, they suffer from cost issues, as conducting these surveys are expensive due to the need for a very large testing pool to get the most accurate results. While expert evaluations provide good insight, they can differ greatly from the PQ of the customer. It is also an issue that these methods often lead to a reactive response, where changes have to be assessed after the product has been made, leading to large costs. Making changes so late in the design process usually requires in a significant financial investment.

Right now, this is quite an unexplored field, and most AI solutions have been focused more on optimizing the manufacturing process through predictive maintenance, detecting defects, and providing tools to assist designers. The work described in [3] is an example of this, demonstrating how AI can improve efficiency in production rather than directly addressing PQ assessment. However, there has been limited research on leveraging AI for real-time PQ evaluation, leaving a gap in understanding how design choices impact customer perception before manufacturing begins. The idea is that is should be used early in the design processes of new cars and in case of a prototype or existing model the images can give a clue on how important it is to fix the design error correlated to the cost.

## 1.2 Purpose and Goal

The purpose of this project is to provide a tool for automotive companies to assess PQ of the customers in the early stages of vehicle design. This can be to analyze an image and give a measurement of how important the error is to fix (cost compared with importance) or a string with a question query that the agent provide answer on the PQ value of that specific customer profile making it able to understand how a design feature might be perceived before implementation. By utilizing this

tool, automotive companies can reduce costs by minimizing the need for extensive customer surveys and other resource-intensive evaluation methods. This allows for a more efficient design process, as real-time insights replace costly and time-consuming traditional assessment techniques. It will also allow engineers to make proactive design choices instead of reactive ones, leading to a more cost-efficient process while ensuring that design decisions align with customer expectations before production begins. The purpose is also to provide an analysis of the trade-offs between different factors like resource utilization, latency, and accuracy, offering insights into how these factors impact the performance and feasibility of AI-driven PQ assessments.

The ultimate goal of this project is to design the back-end technology for an AI-powered tool in which the user should be able to ask questions and receive context-aware responses based on the PQ data. The system will be designed using RAG, which will improve the retrieval and generation of insights from the dataset, and by utilizing LLMs, this will allow the database to be continuously updated. This will provide real-time, data-driven insights that will help the user make informed design choices early in the development stage. The system will be optimized for efficient text retrieval, ensuring that the responses are accurate and relevant to the user's query.

## 1.3 Thesis Outline

This thesis is structured as follows:

- **Chapter 2: Technical Background** describes the technical background of the projects providing valuable insights into key concepts, and technologies relevant to the project. It will go into brief detail about things like Vector Database, RAG pipeline, and LLMs. This chapter serves as a foundation for understanding how these technologies integrate into the system and contribute to improving PQ assessment in automotive design.
- **Chapter 3: Methodology** describes the technical approach used in developing the AI-powered PQ assessment tool. This includes the selection of machine learning models, data preprocessing techniques, system architecture, and implementation strategies.
- **Chapter 4: Implementation** details the practical aspects of system development, including database integration, API design, and the deployment of RAG-based retrieval mechanisms. It also discusses how the system processes and generates context-aware PQ insights.
- **Chapter 5: Evaluation** presents the criteria and methodologies used to assess the system's performance. It includes benchmarking results, comparisons with traditional PQ evaluation methods, and an analysis of trade-offs between resource utilization, latency, and accuracy.
- **Chapter 6: Results and Discussion** analyzes the outcomes of the project, interpreting the effectiveness of the proposed system. It discusses the implications of the results and potential areas for improvement.

- **Chapter 7: Conclusion and Future Work** summarizes the key findings of the thesis, outlines its contributions, and suggests directions for further development and optimization of AI-driven PQ assessment tools.

# 2

# Technical Background

In this chapter, the theoretical technical background necessary to understand the basic fundamental concepts of the subparts of the project is described, starting with vector databases, continuing with the RAG pipeline, and ending with the concepts of APIs, in the same order as the project outline.

## 2.1 Vector Database

A vector database is a database that stores vectors of information. The information could be in the formats of strings, categories, numbers, and more. A vector database stores vectors with similar meanings close to each other. Vector databases are good at capturing the meaning of a sentence rather than specific words [4]. The vectors are stored in high-dimensional space, and it is possible to search for vectors based on a query. This can be illustrated in the example below, demonstrating how three sentences—two of which are closely related and the third not—are stored in a vector database.

- Sentence 1: "Driver drives a Ford car" → [0.25, 0.7, 0.2, 0.1, 0.5]
- Sentence 2: "He always drives a Ford car" → [0.26, 0.68, 0.21, 0.1, 0.52]
- Sentence 3: "Bananas are yellow" → [0.6, 0.1, 0.7, 0.3, 0.9]

These sentences are vectorized into 5-dimensional vectors. The first two sentences have similar vector representations, while the third sentence is quite different. The vector representation of the sentences in a 5-dimensional space is reduced to a 3-dimensional space for visualization, as it is difficult to visualize higher dimensions.

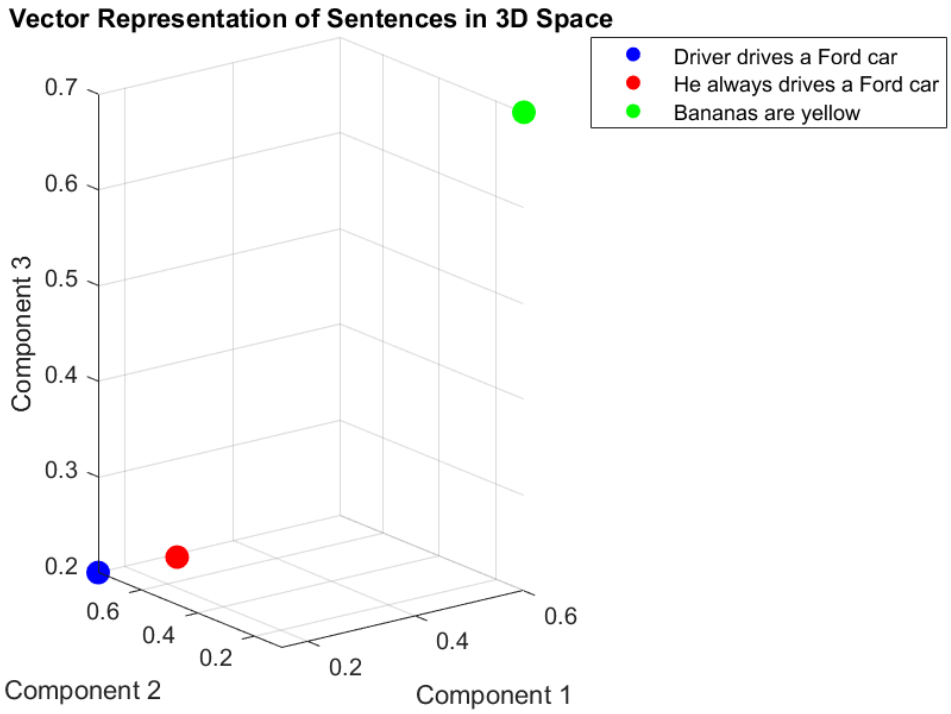



**Figure 2.1:** The vector representation of example vectors shows that vectors with similar meanings are close to each other, while vectors with different meanings have a larger distance between them.

As figure 2.1 shows, vectors with similar meanings are stored close to each other, while vectors with different meanings are farther apart. This technique can be used to search for vectors based on a query. There are different techniques for searching, such as linear or angular search. The first is limited to the boundaries, and points far away could be at the same distance from closer points if the query is near the boundary of the vector database. The angular cosine search, on the other hand, prevents this by also taking into account the angle between the data vectors in the database.

The cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$, is defined as:

$$\text{cosine\_similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \tag{2.1}$$

Equation 2.1 computes the cosine of the angle between the vectors, producing a similarity score ranging from $-1$ (opposite direction) to 1 (same direction), with 0 indicating orthogonality (no similarity) [5].

## 2.2 Latency and Resource Utilization

Latency refers to the time delay between an action and the corresponding response. The action can originate from either the user or the system. In real-time applica-

tions, it is crucial that latency remains low to meet strict time constraints. High latency can lead to degraded user experience, system inefficiencies, or even failures in time-sensitive environments such as autonomous vehicles or medical devices.

However, minimizing latency often comes at a cost. Reducing response times may require increased computational power, more memory, or higher data throughput, which in turn leads to greater requirements. For instance, achieving low latency might involve using faster processors, parallel execution, or dedicated hardware accelerators — all of which increase power consumption and heat generation.

Resource utilization refers to how a system uses its available computing resources — such as CPU, memory, disk I/O, and network bandwidth. High resource utilization can signal efficiency, but it may also indicate resource conflicts. The ideal system balances latency and resource utilization: low enough latency to meet performance requirements without overloading the hardware. This trade-off must be carefully managed during system design, especially in embedded or battery-constrained environments.

## 2.3 Machine Learning

Machine Learning (ML) in the context of Artificial Intelligence refers to different techniques used to teach the computer to perform a task. The learning can be both supervised and unsupervised. Supervised learning means that the computer is trained to, for example, classify objects, where the objects have labels set by humans. Unsupervised learning, on the other hand, can be used for tasks like clustering, meaning that the computer groups objects based on similarity, with the difference that the objects have no labels. ML models can also be used for various regression tasks where the relationship is unknown. Artificial Neural Networks (ANN) is a method that can be used for both regression and classification, and its functionality is described next [6].

### 2.3.1 Artificial Neural Network

ANNs can have different structures depending on the task, but common to all networks is that they consist of McCulloch neurons. A McCulloch neuron in the context of a network is a node that performs a weighted summation of the output of the neurons in the previous layers; for the first layer, it is the input to the network. After the weighted summation, a bias is added, and then the result is fed into an activation function [7]. Below is an example of the structure of a small neural network.

In Figure 2.2, each neuron in previous layer is connected to all neurons in the next layer. Each neuron performs a weighted summation of the input. The output neuron is a single neuron in the figure, meaning that it could be a binary classification task or a regression task with a single output. The number of output neurons is determined by the number of classes. For example, a network that classifies three objects requires three neurons; the exception is binary classification, which can have either one or two neurons depending on how the network is constructed. The number of input neurons is the same as the number of features in the input data. The number of

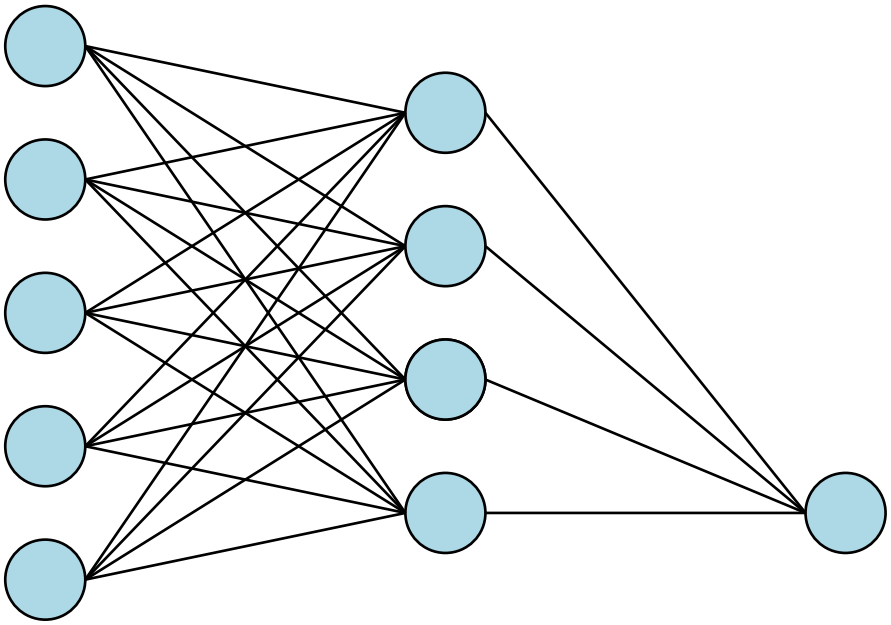


**Figure 2.2:** ANN with McCulloch neurons in blue and corresponding connections from previous layers to subequent layers.

hidden layers and neurons in the hidden layers can be adjusted, and normally, when a network is trained, the parameters are adjusted through several training iterations to determine what is best for a specific problem.

#### 2.3.1.1 Activation functions

As mentioned earlier, each layer has an activation function. There are several activation functions that can be used in an ANN [6].

- **Linear**: Meaning that $f(x) = x$
- **Sigmoid**: A value between 0 and 1.
- **Tanh**: Rescales to a value between $\pm 1$.
- **ReLU**: $f(x) = \max(0, x)$
- **Softmax**: Used in multiclass classification output layers to maximize the likelihood of the correct class while minimizing others, resulting in an array of values between 0 and 1.

#### 2.3.1.2 Loss function

All ANNs have a loss function, which is a measurement of how well the prediction matches the expected output. Below are some of the most common loss functions.

- **Mean Squared Error (MSE) Loss**

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

Where: $y_i$ is the true output value for the $i$-th sample, $\hat{y}_i$ is the predicted output value for the $i$-th sample, $n$ is the number of samples.

- **Binary Cross-Entropy Loss**

$$L_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{2.3}$$

Where: $y_i \in \{0, 1\}$ is the true binary label. $\hat{y}_i$ is the predicted probability of the positive class.

- **Categorical Cross-Entropy Loss**

$$L_{\text{CCE}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{ic} \log(\hat{y}_{ic}) \tag{2.4}$$

Where: $y_{ic}$ is the true label in one-hot encoding for class $c$ of sample $i$. $\hat{y}_{ic}$ is the predicted probability of class $c$ for sample $i$. $C$ is the total number of classes.

- **Mean Absolute Error (MAE) Loss**

$$L_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.5}$$

Where: $|y_i - \hat{y}_i|$ is the absolute error between true and predicted values.

#### 2.3.1.3 Training process

The network can be trained using an algorithm called backpropagation, meaning that the error at the output is propagated backward through each layer. How much each single weight is updated is determined by the gradient, which shows how much each individual weight affects the difference in the outcome [8].

The first step in training is the feed-forward pass through the network, passing through all neurons. First, the weighted sum is computed in all neurons according to equation 2.6. After the weighted sum, the activation function is applied in the layer according to equation 2.7. This is the forward part of the network, and the last layer now contains the output from the network. This is the part that is used when the network has been trained and inserted into the final system.

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)} \tag{2.6}$$

$$a^{(l)} = f(z^{(l)}) \tag{2.7}$$

To update the weights and biases in the network, the following equations are used and described in detail.

$$L = \frac{1}{2}\sum_{i}(y_i - \hat{y}_i)^2 \tag{2.8}$$

$$\delta^{(L)} = \frac{\partial L}{\partial a^{(L)}} \cdot f'(z^{(L)}) \tag{2.9}$$

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \cdot f'(z^{(l)}) \tag{2.10}$$

$$\Delta W^{(l)} = \eta \cdot \delta^{(l)} (a^{(l-1)})^T \tag{2.11}$$

$$\Delta b^{(l)} = \eta \cdot \delta^{(l)} \tag{2.12}$$

$$W^{(l)} = W^{(l)} - \Delta W^{(l)} \tag{2.13}$$

$$b^{(l)} = b^{(l)} - \Delta b^{(l)} \tag{2.14}$$

where $W^{(l)}$ and $b^{(l)}$ are the weights and biases at layer $l$, $a^{(l)}$ is the activation at layer $l$, $z^{(l)}$ is the weighted sum at layer $l$, $\delta^{(l)}$ is the error at layer $l$ and $\eta$ is the learning rate.

As described earlier, the weights and biases are updated according to the derivative for each layer. The loss function is calculated, which is a measurement of how well the output agrees with the correct value. For a regression problem, the Mean Squared Error (MSE), equation 2.8 is typically used, while cross-entropy loss is used for multi-class classification. The first weights to be updated are those in the last layer of the model (closest to the output), since they directly affect the output. These weights and biases are adjusted first during training. Here, the chain rule is used, meaning that the derivative of the loss function is multiplied by the error of the activation function in the layer, according to equation 2.9. Then, for the output layer, the weights and biases are updated using equations 2.11, 2.12, 2.13 and 2.14. For the layers further back, the error is calculated for each specific layer using the same equations. Equation 2.10 propagates the error from the current layer to the layer one step back. This process continues until all parameters are updated.

#### 2.3.1.4 Overfitting

The risk of increasing the number of neurons while having a small dataset for training is that the model may become overfitted, meaning that it learns the input data rather than generalizing, which is not desirable [9]. Below is a list of remedies to prevent overfitting during training.

- **Extended dataset**: More data points generally help the network find relationships rather than memorizing the inputs.

- **Dropout**: This means that random neurons are temporarily disconnected during training, meaning that their output is set to zero, independent of the input. This helps the network generalize since the same input comes with variations due to the randomization in dropout.
- **Pruning**: This is similar to dropout, with the difference that it does not happen randomly. Instead, neurons that have less impact on the outcome (i.e., the output of the neuron is close to zero) are permanently removed from the network.
- **Early stopping**: This means that after each training iteration, a separate test set is used to validate the accuracy of the model. If the accuracy does not increase on the test set for a certain number of epochs, the training is terminated prematurely.
- **Adding random noise**: If the data is relatively clean, it can help to add *small* noise to introduce more variation in the dataset used for training. It is important that the noise is small [10].

#### 2.3.1.5 Vanishing and exploding gradients

As can be seen in equations 2.10 and 2.11, for each layer, the layer's derivative is multiplied by a learning rate. For each layer, the derivative becomes smaller and smaller. On the other hand, if the learning rate is too high, the gradient starts to become large. This is a problem because the last layers are properly updated, while the first layers may not be updated much, especially in networks with several layers. To solve this problem, the same methods used for overfitting can be applied. Using ReLU as an activation function can also help to avoid vanishing gradients, and having a smaller model with fewer neurons in each layer can also be beneficial.

#### 2.3.1.6 Benefits / Transformations to higher dimensions

The benefits of using an ANN are many. It can be used for different types of AI problems, and in the case of complex data, it transforms the data to higher dimensions and find relationships in the data. This transformation is automatically learned through training, meaning it does not need to be specified by the human user [11].

### 2.3.2 Convolution Neural Network

Convolutional layers in a network mean that it has several kernels in each layer, which are swept over an image to recognize features [12]. For example, early layers may detect simple features like a line, while later layers recognize more specialized features.

Figure 2.3 shows an example feature map and how the kernel given in the second column is applied to the upper left corner, with summation performed and the result placed in a matrix. The example shows element-wise multiplication with the kernel applied to a specific part of the feature map, and then the results are summed. The kernel is then shifted one step to the right, and the same procedure is performed

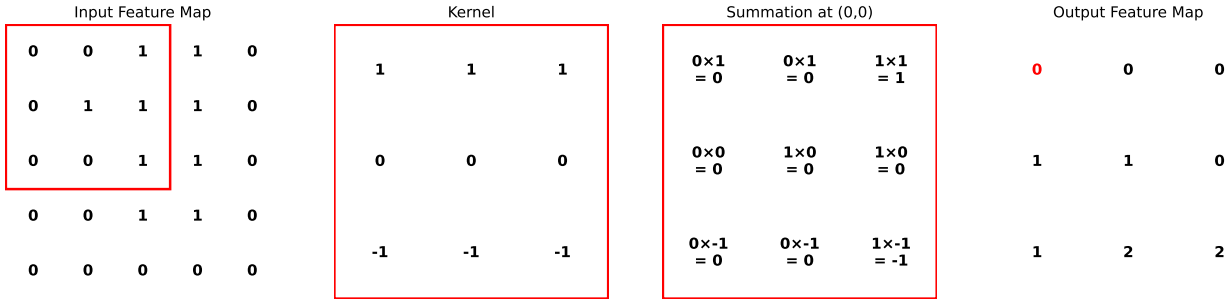



**Figure 2.3:** Illustration of a convolution layer

again, becoming the element (0,1) in the output feature map, and so on until the final output feature map is filled. The step by which the feature map is shifted is specified by the user.

Convolutional networks can also have pooling layers, which work similarly to the convolution layers, but with the difference that the maximum value inside a specific region is saved. Similar to ANNs, CNN layers also have activation functions that are applied to the output feature map before it is forwarded to the next layer. A layer can consist of several kernels, and after the convolution layers, a fully connected layer with an ANN is typically inserted. The training process is the same as for ANNs, with the difference that it is the weights inside the kernel that are updated.

## 2.4 Large Language Model(LLM)

An LLM is a type of machine learning model that has cognition and language comprehension capabilities, meaning it can understand, interpret, generate words, and even respond in human language which is most commonly used in applications like chatbots, and customer service. Some of the biggest and most well-known LLMs that are used today are GPT, LLaMA, and PaLM. These models are trained on a vast amount of linguistic data which makes the model learn language patterns and some subtleties that occur in different languages, which helps in formulating more accurate and more relevant responses.

LLMs are trained using self-supervised learning, predicting missing or next words from large text datasets. The foundation of almost all LLMs in the modern era is transformer architecture, which uses self-attention mechanisms to capture relationships between words in a sentence, regardless of their position, allowing for more accurate context understanding [13]. Unlike previous models, transformers process data in parallel, making them more efficient and capable of handling large datasets. Some common issues that can arise with LLMs are hallucinations, where the model generates information that is factually incorrect or fabricated, biases where any small bias that might be present in the linguistic data can be amplified, and overfitting as above.

As fine-tuning LLMs is very time-consuming, costly, and resource-intensive, prompt engineering has become a crucial technique. Prompt engineering involves designing input prompts for the LLMs which guide the LLMs to generate a more accurate,

and relevant response based on the desired task. This technique can help mitigate issues some of the issues as it ensures that the LLM is directed to providing more reliable and factually accurate responses [14].

## 2.5 Application Program Interfaces

An API is a way to connect computers or different software applications, making them able to communicate, and share data or functionality. It acts as kind of a bridge that connects the two programs allowing for one system to request services or data from the other system. The API is built on well-defined protocols, and rules, which specify how these communications should occur. Making it possible for developers to connect two systems without having full knowledge of the inner workings of each independent system [15]. An example of where an API is commonly used is for GPT which is an LLM. Instead of having to integrate the application into the already established LLM, it can be connected and interact with the LLM through an API. By sending the provided input prompt through the application to the LLM, the application can provide an output response generated by the models. This makes it possible for the application to leverage the capabilities of the LLM, such as text generation and natural language processing, without having to manage or implement the complex machine-learning architecture that makes up an LLM.

## 2.6 Retrieval-Augmented Generation

A Rretrieval-Augmented Generation (RAG) system is an advanced framework that improves the performance of LLMs by the use of external retrieval mechanisms. This design is efficient in providing accurate information as well as improving contextual relevance by pulling in real-time data from external knowledge bases.

The RAG system is separated into two parts: the retriever part and the generation part. The retriever part is responsible for searching and retrieving relevant data, which is stored in vector databases as embeddings, which are numerical representations of data. This gives the model access to the newest data sets and allows it to continuously be updated with up-to-date data, allowing for a more dynamic system. So when the user inputs a query, the retriever converts it to an embedding, performs a similarity search in the vector database, and picks the most relevant information, which is then appended with the original query and sent into the generator. The generator then processes this information and sends this to an LLM. This connection is facilitated by utilizing an API, which generates a response for the user [16].

Compared to more traditional LLMs that rely upon static data, the RAG system works more dynamically by incorporating an external database that is continuously updated with new data. An API is utilized to facilitate the interaction between the RAG system, the external database, and the LLM. This leads to a response that is much more accurate and provides up-to-date responses. It also reduces hallucinations, which is a common issue in LLMs where it generates a response with fabricated information [17].

# 3

# Methods

In this chapter, we give a brief outline of the work method. The first step involves technical background research to understand the components used in the project. This includes studying the datasets, which are divided into two parts: one containing images and the other containing customer attributes. Understanding the structure of each dataset and gathering relevant background knowledge is essential to making informed decisions about the necessary parts of the system. Additionally, machine learning methods and libraries are researched to identify the most suitable ones for the task.

Following the prestudy phase, the implementation process begins. The first task is to preprocess and structure the dataset to ensure the data is accurate and consistent. This will be followed by the creation of a vector database, which would serve as the core for data storage and retrieval. After the database, the next step involves implementing machine learning methods, including an RAG pipeline, to connect and utilize it for generating useful results. Finally, the last phase involves integrating the system with an API to enable communication between the backend and frontend. The tool used to write code is Deepnote, GPT will be used as LLM and Fly.IO for API integration and Google sites for frontend ebsite.

Throughout the implementation process, the performance of the prototype is continuously evaluated, particularly in terms of latency, memory usage, and speed. Adjustments are made in real-time to optimize the system based on these parameters. Latency reports are compiled for different implementations, and the best solution is ultimately chosen based on the expected use and needs of the system.

# 4

# Implementation

In this chapter, an overview of the implementations for the different components will be provided, along with the techniques used to develop them. Each component represents a key part of the overall final system.

## 4.1 Vector database

The first process of using the vector database was preprocessing available data. There were a data set consisting of customers' PQ ratings and demographic information, such as income, current vehicle, and education, and their corresponding PQ value ratings based on attributes like gap, paint execution, and surface finish. This dataset was not gathered during the project but was already available. These are used to predict and determine PQ ratings.

Text data, such as current car and education, were mapped to vector embeddings via a sentence transformer model. The all-MiniLM-L6-v2 sentence transformer [18] was used since it balances between efficiency and the quality of embeddings. This mapping helps capture semantic relations between attributes and their significance in estimating PQ values. These generated embeddings are further normalized by FAISS [19] to allow for efficient computation of similarities, specifically cosine similarity, which was needed for retrieval and clustering operations in the vector database. Our company supervisor recommended using FAISS. Other indexing methods like Hierarchical Navigable Small World were briefly explored, but FAISS was found to be the most efficient and most straightforward to implement.

Lastly, the text embeddings are combined with the numerical features to create a single vector representation for each customer. To enhance retrieval efficiency, an index was built to store the vector representations. Indexing enables efficient similarity searches, which allows the system to retrieve the most similar customer profiles for a query. The vector representation were then used to train a machine learning model for PQ value prediction based on customer features.

## 4.2 ML Model to Predict PQ Insights

The machine learning model for predicting PQ insights was designed to leverage the structured vector representations generated from the vector database. By utilizing both numerical attributes and textual embeddings, the model aims to learn

complex relationships between customer demographics, vehicle attributes, and PQ ratings. A neural network was implemented due to its capability to model non-linear dependencies and capture intricate patterns in the data.

To enable the reuse and deployment of the trained model, functions for saving and loading model weights and biases were implemented. These functions store the trained variables in files, ensuring that the model can be used without retraining from scratch. The training process itself was carried out using the Keras TensorFlow library [20], following a structured approach where the model architecture was first defined, parameters were specified, and the model was subsequently trained using the fit function.

Due to the nature of weight initialization, the results from training may vary across different runs. To mitigate this, multiple training sessions were performed, testing various configurations and model architectures to determine the optimal solution. This iterative process allowed for fine-tuning of the model to achieve the best predictive performance.

A neural network architecture was selected for the prediction task, designed to learn complex relationships between customer attributes and PQ ratings. The model consists of multiple fully connected layers, with ReLU activation functions applied to introduce non-linearity and avoid problem with vanishing gradients, explained in Section 2.3.1.5. The initial layer processes the combined vector representation of customer data, followed by several hidden layers optimized to capture relevant patterns. To improve generalization and prevent overfitting, techniques such as early stopping and learning rate scheduling were applied. The Adam optimizer [21], with gradient clipping, was used to ensure stable convergence during training.

The final trained model was evaluated on a hold-out test set to assess its performance. Mean squared error (MSE) was used as the loss function, while mean absolute error (MAE) provided an additional metric for interpretability. Once validated, the trained model was saved and could be deployed to generate PQ predictions for new customer inputs, providing valuable insights for the system.

## 4.3 Model for Image analysis

The image model contains defects of the cars, and a value should be predicted for both the perceived quality and craftsmanship score, which reflects how an expert would rate the perceived quality compared to a normal customer.

### 4.3.1 Pre-processing

The images that were sent to the network were colored images, and some had a circled error while others showed the raw error for the same issue. The vehicles and color contrast around the error differed, but the optimal solution was to identify the error regardless of the car's or interior's color. To solve this problem, all images were converted into grayscale. Next, to ensure the defect in the images was recognized independently of the position and angle, the images were copied into several batches and randomly rotated, horizontally and vertically flipped, and zoomed.

### 4.3.2 Training

To be able to save and use the model later, functions for saving and loading the weights and biases (the variables) from the network were constructed. These are simple functions that save the variables into files. For training the model, functions from the library Keras TensorFlow [20] were used. First, the model architecture was defined, and then the hyperparameters were specified before the model was trained by calling the fit function. Some randomization occurs due to the initialization of the weight matrices, so the results from training can differ from run to run. Therefore, several runs were performed, and different hyperparameters and architectures were tested to find the optimal solution.

CNN was used, and several different models were built together to achieve the best possible performance. First, a base model was trained. This model predicts the area of the car, such as the front, back, sides, instrument panel, etc. Based on the area, a specialized model was trained for each of these areas to get the best outcome. The first model has an array output with the percentage likelihood that the image belongs to a specific class, while the specialized models have two regression outputs: perceived quality and craftsmanship score. The base model consists of three convolution layers, all with max pooling included, and three fully connected layers (ANN), including the output layer.

In the specialized models, the number of convolution layers was six, with the first three identical to the ones in the base model. These layers are frozen, meaning their weights are not updated during backpropagation. The benefit of this was that the pre-trained model could identify the same features as the base model, and based on that, continue to find relationships from the input data in combination with the outcome from the first base model. Additionally, there are three ANN layers, including the output layer, which has two neurons—one for the perceived quality and one for the craftsmanship score.

Different parameters and architectures were tested, and the model was rerun to obtain the best result.

## 4.4 RAG pipeline

The RAG pipeline was a concatenation of a static prompt with dynamic information variables from the vector database in the project. The prompt was then sent to ChatGPT to build a text-based answer on the user's question. The RAG pipeline consists of three main functions. The first, *retrieve*, fetches relevant information from the user query and the neural network based on the insights in the string. The next function, *create_prompt*, creates a text-based string from the insights together with the dynamic information fetched from the *retrieve* function. The third function, *generate_response*, sends the generated prompt to ChatGPT via API. There are some settings that are adjustable in this function: the specific model, the maximum number of tokens in the answer, as well as the temperature, which determines how much randomization the answer should contain.

The flowchart block diagram for the pipeline can be seen in Figure 4.1. As described

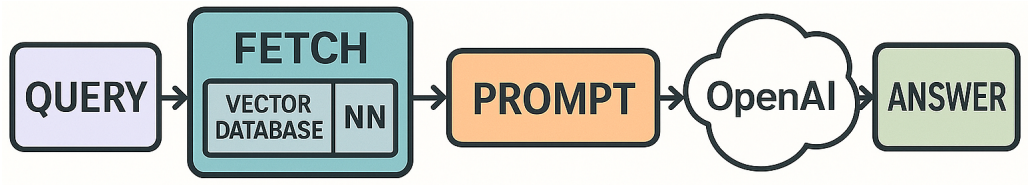



**Figure 4.1:** Block diagram of RAG pipeline flowshart

in the section before and in Figure 4.1, the query is entered, then data is fetched, the prompt is created, sent to ChatGPT via API, and an answer is received.

The goal of the prompt was that, together with the information given from the vector database and neural network and the static part of the prompt, it generates an answer that is relevant and reliable, meaning that the model should not hallucinate—for example, add attributes that do not exist or say something that is totally not relevant to the user's question. ChatGPT often, when it is unsure, adds attributes. For example, if the user's question is about spatial harmony and the information in the database is weak, ChatGPT could include tires and oil. This is not what the system wants, so therefore the prompt is very important.

The prompt consists of the information given from the vector database and predictions through the neural network, the temperature was set to 0.7 as well as specific instructions to ChatGPT:

- The user query can contain a question or information like a specific driver profile, kind of car, miles per year, etc.
- Provide a short and accurate answer to the user's input/question based only on the information above.
- Do not add any new attributes or external information.
- Only analyze what's given from the database and predictions, and relate it to the user's input.
- If the user input is not relevant to the insights and predicted values, or to the perceived quality of cars, say so in a polite way without referencing unrelated topics.
- Use the importance values PQ (1–20) to prioritize, where higher means more important. Do not mention these limits.
- Do not write the answer in first person or third person with pronouns.

## 4.5 API implementation

This project uses two different APIs: one API connects the vector database in the backend to an LLM for more advanced query processing and in order to get a more human-like response, and the second API connects the backend to the front end, allowing for communication between the two components.

### 4.5.1 API for LLM

The first API was used for the ML model to predict PQ insights, which was utilized by connecting the vector database to an LLM. This was done in order to generate human-like responses based on the available vector database data. It also allows for more complex input queries as the LLM has a easier time at interpreting and extracting the relevant data from the input query, giving the ability for the ability to interact with the system in a more natural way as well as being given more human-like responses.

The LLM that was used in this project was OpenAI's GPT-4.1, which was chosen for its strong performance and for its ability to understand the context of user queries, extract relevant information from the vector database, and provide coherent, accurate, and conversational responses. The downside of using GPT was that it cannot be launched locally, so it has to be launched through OpenAI's servers, which introduces problems in regards to latency response time. GPT also requires a subscription, so there are some costs associated with it, specifically when dealing with very large queries, which consume a lot of tokens. But it was still chosen over free options due to safety concerns, as OpenAI's models have undergone extensive testing and moderation to reduce harmful or biased outputs, making them more reliable.

### 4.5.2 API for Connecting Backend and Front-end

The second API used in this project was used to connect the backend logic, which includes all the different models, to the front-end interface, which allows for communication between the two different components and allowing for real-time user interaction. The API used for this connection is FastAPI, a common web framework used for building APIs for Python. This API was chosen due to its simplicity and speed, which allowed for a simple implementation and provides a low latency response time. FastAPI works by using HTTP requests to communicate between the front-end and back-end. It defines endpoints that handle these requests, such as GET or POST, allowing the front-end to send data to the back-end models and receive responses in real-time. Thanks to its asynchronous capabilities, FastAPI can efficiently manage multiple simultaneous requests, ensuring fast and reliable interaction.

## 4.6 Front-end implementation

As mentioned earlier, a part of the project was to connect the backend logic with a front-end interface. This front-end was originally supposed to be designed by a separate group, but because their project aim changed from practical to a more theoretically based project, a prototype front-end was designed for this project just to make sure all of the connections worked as expected. The front-end was implemented using HTML and JavaScript and was designed as just a very simple prototype. Therefore, the front-end website was designed with minimal styling and

a lack of interactivity, just focusing on the raw and most essential parts, like being able to retrieve and deliver a text prompt and the ability to send pictures.

## 4.7 Latency analysis and resource usage

Throughout this project, two main aspects of system performance are analyzed and measured: latency and resource usage. The focus was on achieving fast response times while minimizing the use of computational resources.

Latency analysis involves identifying bottlenecks in the system. This includes delays in data processing and slow access to similar elements stored in the vector database. Reducing latency was especially important for real-time applications, where responsiveness was a key requirement. To address this, different techniques are tested and refined, including batch request handling, optimization of retrieval methods, and improvements in indexing efficiency.

Resource utilization focuses on how effectively the system leverages available hardware, such as CPU, GPU, memory, and storage. The objective was to improve efficiency without placing unnecessary load on the hardware. Methods such as parallel processing, GPU acceleration for compute-intensive tasks, and better memory management are applied to improve overall performance.

By continuously tracking both latency and resource usage throughout development, the system was progressively optimized. This ensures not only accurate and efficient results, but also the ability to scale with growing data volumes and increased user activity.

# 5

# Results

In this chapter the results from the project are presented, including the vector database, the different ML models, RAG, API, front end, resource utilization and latency.

## 5.1 Vector database

The result of the use of the vector database was positive, in that it correctly retrieved the closest matches using the cosine similarity measure. By converting text-based and numeric-based customer information into vectors, the system was able to successfully retrieve similar customer profiles. This is crucial in predicting PQ values according to similar attributes such as demographic information and customer interests.

Indexing the vector embeddings with FAISS significantly improved retrieval efficiency, resulting in faster predictions for PQ values. The system was able to retrieve relevant customer profiles almost in real-time, demonstrating the effectiveness of the vector database at handling data efficiently.

Further tests confirmed that the retrieval accuracy was high, and the system was always fetching the most similar customer profiles from the input query. This not only enhanced the performance of the model to predict PQ ratings but also showcased the ability of the vector database to enable both clustering and retrieval operations.

## 5.2 ML model to predict PQ insights

The result of the ML model proved to be quite accurate if the input query included all of the different attributes. So if a user inputs a query with identical attributes as one already available in the dataset, the prediction for the PQ values is around 90-95% accurate. The model also shows promising results when some of the attributes are missing. This was because the dataset was duplicated and expanded in order to get more data to train with, and in this expanded dataset, some of the attributes are blank, so if the input query into the system is blank, it will show a result that seems quite reasonable. However, the accuracy tends to decrease as more attributes are missing, since the model relies on the available data patterns to make predictions. Despite this, the model demonstrates robustness in handling incomplete queries by leveraging learned correlations from the expanded dataset. This suggests that

the system can still provide meaningful predictions even when some attributes are unavailable.

## 5.3 Model for image analysis

The result for the image analysis model is that it seems to be working well on images where the error is marked, but less effectively on areas that are more general. Such as on images of just a door without the small error marked. In Table 5.1, some key performance metrics are stated.

| Metric | Value |
|---|---|
| MAE_pq | 5.7481 |
| MAE_cpq | 0.7938 |
| RMSE_pq | 8.5198 |
| RMSE_cpq | 1.3539 |
| MPE_pq (%) | -26.1929 |
| MPE_cpq (%) | 3.9401 |
| STD_pq | 8.4976 |
| STD_cpq | 1.3136 |

**Table 5.1:** Performance metrics for the model.

The table presents various performance metrics used to evaluate the model.

- **MAE (Mean Absolute Error)** measures the average magnitude of the errors in the predictions. Lower values indicate better performance.
- **RMSE (Root Mean Squared Error)** provides a measure of how much the model's predictions deviate from the actual values, with higher values indicating worse performance.
- **MPE (Mean Percentage Error)** represents the percentage difference between predicted and actual values. A negative value means the model tends to underpredict, while a positive value suggests overprediction.
- **STD (Standard Deviation)** gives an indication of the spread or variability of prediction errors. Larger values suggest higher variability in predictions.

From the table, it is clear that the model performs significantly better on CPQ images. The low MAE and RMSE values (0.7938 and 1.3539) indicate that the model is both accurate and consistent in these cases, with a small overprediction bias (MPE 3.94%) and low variability (STD 1.3136). In contrast, for PQ images, the performance drops notably: higher error values (MAE 5.7481, RMSE 8.5198), a large underprediction bias (MPE -26.19%), and greater variability (STD 8.4976). This suggests that while the model is effective in targeted error detection, it struggles to generalize to less specific image contexts.

The values represented in Table 5.1 are from the test set, meaning that they were not used during training. The Gaussian distribution of the error is shown in Figure 5.1.

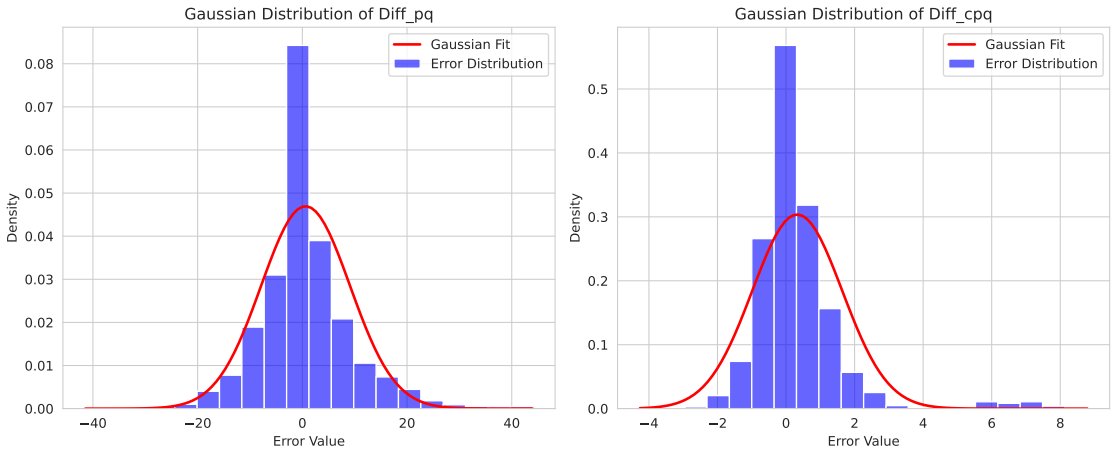



**Figure 5.1:** Gauss distribution of prediction

Figure 5.1 shows the distribution of the error with respect to standard deviation for PQ values on the left and craftsmanship score on the right. The bars represent the percentage (y-axis) of values at each standard deviation (x-axis) from the mean. For PQ, the distribution is symmetrical, while for craftsmanship score, there are more samples that deviate towards the upper part of the Gaussian distribution, meaning that the model can predict to high values sometimes for the CPQ.

## 5.4 RAG pipeline

The response generated from ChatGPT with the RAG system summarizes the relevant insights given from the database, connects it to the user's question, and gives an answer that is relevant. Since the temperature variable was set to 0.7, meaning a bit of variation in the answers, if the user inputs the same query, it will receive a bit of variation in the answer. The variation in the answer is in how it is presented to the user—the main information about the PQ insights and conclusions is still the same, but how ChatGPT says it is a bit different.

For non-relevant questions or strings, the returned message generated told the user that the question is unclear or not related to the PQ insights of cars. Therefore, the system is focused just on the PQ insights of cars; it's not able to receive an answer on other questions such as how the weather is. The answer does not contain any additional attributes or hallucination or other non-relevant information; it gives a succinct answer. How the instructions was formed was very crucial for the functionality. For each API request to the GPT, around 1500 tokens are consumed, which includes the system prompt, user query, retrieved PQ context, and the generated response, which is a reasonable amount given the level of detail, insight, and accuracy required.

## 5.5 API implementation results

The connection with the first API was successful and enabled effective communication between the vector database and the GPT-4.1 model. This integration allowed the system to handle complex user queries by extracting relevant PQ insights and generating coherent, human-like responses. Throughout testing, the API demonstrated reliable performance with no significant interruptions or errors, confirming stable integration between the components.

The second API successfully connected the backend logic, including the various models, with the front-end user interface. This enabled real-time communication between the components, allowing users to send input and receive responses seamlessly. FastAPI's asynchronous design effectively managed multiple simultaneous requests during testing, maintaining stable and responsive interaction without errors or crashes. The implementation facilitated a smooth user experience and reliable data exchange throughout the project.

## 5.6 Front end implementation results

The front end was designed with a minimalist design, just focusing on the most essential parts. The final design of the website can be observed in Figure 5.2, which includes a:

- **Text prompt** where the user can input their query.
- **Results display box** that presents the most relevant matches retrieved from the vector database based on the user's query.
- **Image upload button** that allows the user to upload images for image analysis.
- **graph** displaying the PQ and CPQ values of the submitted image, giving the user an idea of how severe the defect is.

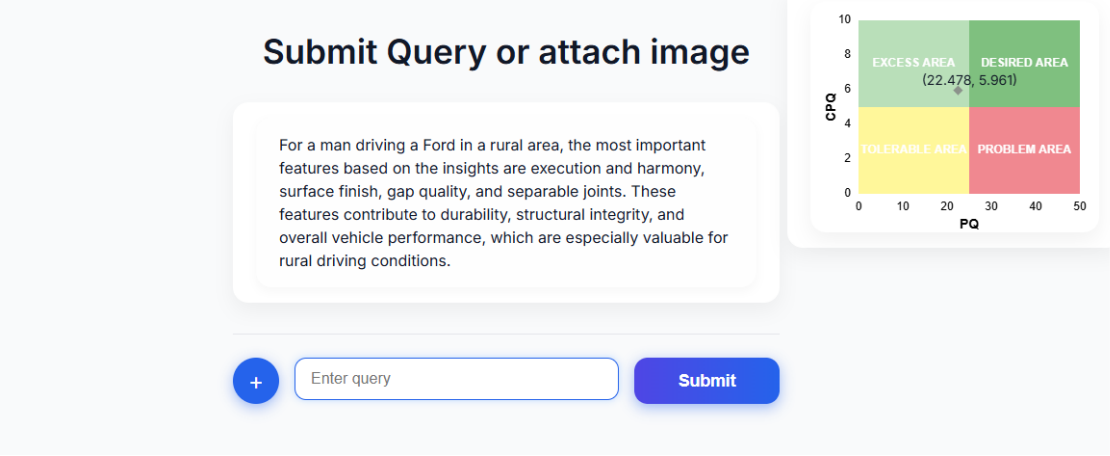



**Figure 5.2:** The website along with its basic features

The text in the text box shows the response from the model to predict PQ insights and vector database using RAG pipeline, the input query was "What are the most important design features for a man living in a rural area driving a Ford less than 500 miles per year?". To the right in Figure 5.2 shows the result from an image returned from model for image analysis, both the PQ value and Craftmanship score (CPQ) is shown in a quadrant.

## 5.7 Latency analysis and resource utilization

Throughout the project, latency and resource utilization measurements have been conducted to evaluate the system's performance and efficiency. These analyses aim to assess the trade-offs between different criteria, such as latency response time, resource usage, and the accuracy of the AI models.

### 5.7.1 Latency analysis on the systems

In this section the result for the Latency of the different parts of the system is presented in detail.

#### 5.7.1.1 Latency analysis on the backend

In this section, the latency analysis, when done on the backend, will be presented. The backend models were run virtually using DeepNote notebook running on the basic free option provided by DeepNote. This machine provides 2 Virtual CPUs and 5 GB of RAM on a shared Intel Xeon server-grade processor. The results of running the different models on this machine is provided below in Table 5.2. The latency for the image analysis is relatively low, as is the latency for querying the vector database without an API connection. However, the latency increases significantly when the vector database query is routed through the GPT-4.1 API. This is primarily due to the additional overhead of communicating with OpenAI's external servers and the processing time required by the LLM. Although the vector database query latency is higher when routed through the API when running it without the LLM, it is still preferred, as with the LLM, it allows for more natural and flexible input queries and gives a better, more natural response in return.

| Model | Latency |
|---|---|
| Image analysis | 889 ms |
| Vector database Query (No LLM) | 92 ms |
| Vector Database Query via GPT-4.1 API | 3.18 s |

**Table 5.2:** Latency for Backend Model Components on Virtual Machine (2 vCPUs, 5 GB RAM)

#### 5.7.1.2 Latency analysis on the front-end

Latency analysis was also done when connecting the backend part with the API to the front-end website. Here, the latency time for the image analysis and the Vector

Database query via the GPT-4.1 API was checked. When running the backend code with the front-end, launched through GitHub using a virtual machine as well, but this virtual machine is more powerful than the one provided by Deepnote. This machine is composed of 4 virtual CPUs, and it has up to 8 GB of RAM. Using this machine will reduce the latency for the back-end tasks like image analysis and the local vector database querying. This can be observed in Table 5.3, where the latency for image analysis decreases on average by approximately 200ms, and the latency for the vector database query drops by around 800ms. What can be noted is that when smaller images are used, the latency time goes down significantly, so when using images the size of 256x256 pixels, the latency can go as low as 80ms. As explained earlier the images are reduced and gray coded before being sent into the code, and if the image is already small and gray coded the program has an easier time sending it into the model compared to the images that are a part of the test dataset, which are significantly bigger around 6848x6848 pixels.

| **Model** | **Latency** |
|---|---|
| Image analysis | 648 ms |
| Vector Database Query via GPT-4.1 API | 2.248 s |

**Table 5.3:** Latency for Backend–Front-end Integration on Virtual Machine (4 vCPUs, 8 GB RAM)

### 5.7.2 Resource utilization and training time for the models

In this section, the resource utilization and the training time for the different models will be analyzed to determine the most efficient model in terms of computational cost and scalability. The result of these tests is shown in Table 5.4. What can be observed is that the base model and the combined models represent the total training time of 11 individual models added together. The computing time it takes to train each model is based on the number of epochs used, as the more epochs used for training a model, the higher the training time. However, beyond a certain point, increasing the number of epochs yields diminishing returns in model performance, resulting in unnecessary computational cost without significant accuracy gains. Therefore, identifying the optimal number of training epochs is essential for achieving a balance between efficiency and performance.

| **Model** | **Average CPU** | **Average RAM Utilization** | **Training Time** |
|---|---|---|---|
| Base Model | 62.82% | 30.75% | 20906s |
| Models combined | 56.66% | 32.56% | 45985s |
| PQ-insight Model | 59.85% | 38.57% | 5001s |

**Table 5.4:** Average CPU/RAM usage and training time for each model

Early stopping was implemented, meaning that when the model does not improve on the testset for a predefined number of epochs, the training is stopped. The machine used also has its limits on the number of epochs it can run before becoming unstable, due to time restrictions in the shared virtual CPU environment, which further

emphasizes the need to select the training duration carefully. Exceeding these limits can cause system slowdowns or crashes, negatively impacting the training process and overall productivity. Thus, optimizing training not only improves resource efficiency. For some of the models, a lower number of epochs can be used without significantly compromising performance. For example, the base model, which is responsible for identifying parts of the car in an image, was trained on a relatively large dataset. Because of this, each epoch required substantial computational resources and time to complete, while some models for ranking the severity of each part could be run using more epochs, as they had less data to be trained on.

The CPU utilization varies between the different models a bit, ranging from around 56% to around 63%, which shows quite stable performance. If the average CPU utilization becomes too high, this can lead to system instability, such as crashes, freezes, or interruptions in the training process. These instabilities can make it so the training process stops, which means that the whole training process has to be restarted, leading to data loss, which is time-consuming and inefficient. Therefore, the training process was optimized by improving resource management, balancing computational load, and minimizing sudden CPU spikes.

In total, the back-end application requires around 4.5 GB of RAM to run; this includes all of the models, the preprocessing pipeline, and the necessary supporting services needed to handle inference requests efficiently, the machine currently used to host the back-end application currently has 8 GB of RAM which is more than enough RAM to be able to host the application.

# 6

# Discussion

In this chapter, the results presented in previous section are explained and discussed.

## 6.1 Vector database

The vector database was implemented to predict the PQ values for the different problem areas more efficiently and accurately. The integration allowed for the text embeddings to be combined with the numerical features in order to make a unified vector representation for each customer profile. This approach enabled the system to capture both semantic and quantitative aspects of the data, which in turn improved the prediction and retrieval processes. The method that was used was FAISS combined with the sentence transformer MiniLM-L6-v2, which proved to provide a very accurate and fast similarity search mechanism for combining and querying customer profiles.

What could have been explored more in the project would have been to fine-tune the sentence transformer model on domain-specific data related to the automotive industry. While MiniLM-L6-v2 offers a good balance between speed and quality, it is a general-purpose model and may not fully capture the nuances of terminology specific to perceived quality in vehicles. Domain-specific fine-tuning could have enhanced the semantic accuracy of the embeddings, potentially improving both retrieval precision and model interpretability. During implementation other vector database models were explored, such as Pinecone and Weaviate, but the result from using other models were less accurate and also in some cases required more resources, such as installation packages and connections through API to a server. So therefore FAISS was the optimal solution for the purpose of our project.

## 6.2 ML model to predict PQ insights

The ML model used to predict PQ insights worked very well and provided good results. The connections with the model with the Vector database worked as planned, and there were no major issues encountered. What could have been done was to have trained the models using more epochs and using a higher-performance machine on Deep Note. The dataset that was provided contained around 2000 customer profiles, which is quite extensive, but having an even larger dataset could have provided even more accurate results. A broader dataset might help the model capture more

nuanced patterns across different customer segments, resulting in more robust predictions, especially for edge cases or underrepresented groups.

When it comes to the decision to use a neural network, several other classification models were tried. Some of these were Logistic Regression [22], Random Forests for regression [23], and CatBoost[24]. These models didn't perform well on the dataset, and we suspect the reason for this is that the nature of the dataset is complex with several outputs (one for each PQ segment). Therefore, the data is complex, non-linear, with multiple outputs for each run. The neural network, as explained earlier, is made up of several layers where each layer has its own output, and within these layers, individual neurons generate activations that contribute to the final prediction. This enables the model to produce predictions that are better suited to the dataset's high-dimensional complexity.

When it comes to the decision about the number of layers, activation functions, and the number of neurons in each layer, different models were tested. Applying fewer layers with the same number of neurons or fewer neurons in the layers resulted in a less accurate model for the PQ predictions. When the number of layers and neurons were increased, on the other hand, the performance became slightly better, but the training time became longer and the resource usage increased. Since the accuracy was only marginally better, the decision fell to the model used.

## 6.3 Model for image analysis

As stated above, the model for image analysis works well on images where the error is marked, while more general images yield less accurate results. This is also reflected by the numbers presented in Table 5.1, where the MPE is -26%, meaning that the mean deviation is lower than the actual predicted value. The MAE is around 5, which is also influenced by the non-marked images. Additionally, it should be considered that the dataset used in this subtask is quite small, which negatively impacts the results. Since the dataset is small, the risk of overfitting is higher. This is problematic because the training has seen fewer variations and examples of the defects. Normally, to get a good and accurate network, thousands of images are needed, but here, only around 100 labeled images are available.

The results for the craftsmanship score (CPQ) in the table are more accurate than the PQ values in some cases, and this is due to the fact that the craftsmanship score is more even and consistent. Many of the data points are centered around the same values with small variations, and the result reflects this. As can be seen in Figure 5.1, the craftsmanship score has more deviation in the upper part of the distribution, meaning that the predicted values tend to be higher than the actual values. One reason for this is that the craftsmanship score values are smaller, while the PQ values are higher with a larger interval. To achieve more accurate results, more images are needed to help the model generalize effectively. It would also be beneficial to obtain images from different car models with a wider range of deviations.

The reason for choosing CNN for the image analysis was that CNN is suitable for image analysis since it uses kernels that are swept over the image, meaning that the

input image can be rotated, flipped, or mirrored, and the network will still find the error. In comparison to using normal ANN, the error needs to be at the exactly same position in the image. Also, in the image analysis, different layouts of the networks were tested. The final version includes a two-stage architecture, but an earlier model consisted of only one network model. The single network gave poor regression on the images, independent of the size (larger network was slightly better but still poor). We think that the reason was that the eleven different categories had such huge span in the PW rating, while at the same time some different errors were slightly similar in the shape. For example, a scratch on the outside paint looked the same as a scratch on the dashboard, and since CNN uses kernels, it could be difficult to tell them apart. Therefore, by training one base model that predicts the error area of the car was implemented, this model focuses not only on the error but also on the environment. So in this model, the model can tell the outside paint apart from the dashboard, for example. By training specialized models, the error for each area could be better adapted to the local PQ value range.

Also, different layer structures were tested. For the base model, three convolution layers and three FC layers give a suitable accuracy; increasing the layers does not improve the result, while fewer layers give a poor result. This architecture was chosen based on the results. The specialized models also tested different models, and the reason that the first three convolution layers are the same as in the base model was that we want the model to keep the feature selection performed by the base model. Therefore, these layers were copied and not updated during training. The rest number and sizes were tested, several models were tested, and one recurring problem was that the accuracy became high for images that are very close to the ones in the dataset. Since the dataset is relatively small, the model became overfitted, meaning it performs well on very similar images and poorly on the rest. Therefore, increasing the number of layers does not improve the result. The fact is that more data is needed to get a generalized model that is not overfitted. But compared with the combined model tested before splitting it up into the two-stage architecture, these models give better predictions of the PQ and CPQ scores.

## 6.4 RAG pipeline

The RAG pipeline system works according to the functionality of fetching data from the vector database and neural network prediction model, creating a prompt, sending it to ChatGPT and receiving a reliable prompt. The results show that the system works pretty well when the user gives a query with much relevant information about the user profile. The vector database performs better, which also affects the whole RAG system. As described earlier in the vector database section, this could be explored further. Currently, the user inputs a query and receives an answer. What could be improved from the prototype to future implementation could be to make the user able to continue the chat with further questions in the same conversation. As the model is built now, the chat restarts when the user enters a new query, i.e., the memory of earlier conversations is cleared.

Further improvements of the RAG system can be to build a custom LLM model

instead of using ChatGPT via OpenAI API. This adjustment will reduce the cost since the OpenAI API service is expensive. If is not much, but if the system should be used by professionals, the long-term cost will be reduced by building the LLM from scratch. This adjustment will also increase the security of the system since no third-party company gets access to the information that is valuable for the company.

As the system works now, it equally states the predictions from the network and the five neighbors from the vector database in the prompt, but depending on the query from the user, the result from the neural network or vector database can be more or less significant. The network predictions work best when the user asks about a specific driver profile, while the vector database gives better results on more general questions. For future implementations, some weight factor between these two can be implemented to get even more reliable results.

Further improvements can also be to connect the image PQ predictions with the text-based, for example, the PQ values from the network model. The user can input a query describing a specific driver profile to get more detailed information about how the specific result is connected to a driver profile, but to make it possible to implement this, more data is needed, including the connections between the defect and the user profile. The data given now lacks that information.

## 6.5 API implementation

The API for the ML model to predict PQ insights worked quite well and the GPT-4.1 provided a good contextual understanding of the user's input query, as well as providing good contextual and relevant responses to the user. An issue that could be brought in with the API would be the amount of time it takes to receive a response. When running the ML model it provides a very fast response time around 89ms, but when being connected to the API it takes around 3s which is higher than the time set in the beginning of the project, but it was still deemed acceptable for practical use. What could have been explored would be to try different LLMs like GPT-3.5 or more lightweight transformer models, which might offer a better trade-off between response time and contextual understanding. Another approach could be to implement caching mechanisms for frequent queries or batch processing of requests to further improve overall system responsiveness.

## 6.6 Front-end

The front-end website that was designed for this project works as a prototype interface for users to interact with the PQ prediction system. It allows users to input relevant customer information and receive predicted PQ insights in a user-friendly manner. While the prototype successfully demonstrates core functionality, including real-time querying and displaying prediction results, there is room for further development. However, as stated earlier, the design and development of the front-end interface were not a part of this project but were originally the aim of another group. This other group focused on researching how designers would prefer the layout and

functionality of such a website. Their research was not taken into account when making the design of this website, as the front-end in this project was built with the primary purpose of trying out the API connections between the front end and back end, ensuring reliable data transmission and system integration. Therefore, as part of future work, the front-end could be redesigned based on the insights gathered by the other group regarding user preferences and interface design.

## 6.7 Latency analysis

As can be seen in Table 5.1, the main time is due to the API connections through OpenAI taking 3s. This is the critical part of the system and this path can be significantly reduced by building a customized LLM model. It was also found that the latency for non-relevant questions was a few milliseconds faster when having the real data. This might be due to the LLM analyzing the real data, while when entering something non-relevant, it doesn't need to make the analysis to find the correlations between the query and fetched data. But here also the program could be changed to just return the insights without the analysis, meaning that users needs to make the analysis of the outcome themselves. This means that there is a trade-off between quality and latency.

The latency report also shows that the latency when running on Fly.IO with more memory and CPU is better than when running on Deepnote, but is expensive to have larger memory and CPU. So even here, it is a trade-off between memory, CPU, latency, and money. The conclusion is that faster programs cost both more money and performance in terms of CPU and memory. Also the training could be sped up at the cost of memory and CPU.

## 6.8 Ethical aspects of the system

This project, like most other projects, comes with ethical challenges when it comes to the pre-study, implementation decisions as well as the final usage.

### 6.8.1 Data collection

The datasets used during training regression models were given from the company at the beginning of the project. One problem is that it's not clear how the data was collected, since it may affect the result. From the ethical perspective, it is important that the user knows how their data will be used; the information during data collection is therefore important. Other questions are how the participants in the dataset were selected (what kind of people were able to attend) and if they were given economic compensation or not for the data.

### 6.8.2 Third party tools

In the backend system, the API connection with OpenAI's ChatGPT is a part of the system and here it is very important to know that it's a third-party tool and they

receive our data—the prompt the user sends and the relevant insights—therefore, from an ethical perspective, it's important to emphasize transparency and limit sensitive information. Important to know here is that it's unknown how ChatGPT is programmed—maybe it is biased or sponsored in some way that may affect the response. This is important to be aware of.

### 6.8.3 System usage

From an ethical perspective, it is also important to know how the system is going to be used and what the consequences are. The purpose of the system is to develop a helping tool for the designers during the design phase of new cars. The system will relieve the workload for the designers, but this can also cause staff reductions. This means that a few people may need to leave their job and find new jobs, and from an ethical perspective, this may be considered.

### 6.8.4 Use of AI in report writing

All content in this report has been written in our own words, based on our understanding and research. We have not copied or plagiarized any material from external sources. However, we have made use of AI-based tools, such as Grammarly, solely for the purpose of grammar and language refinement. These tools were employed to enhance clarity and correctness in writing, without altering the original text.

# 7

# Conclusion

The prototype of the system has the desired functionality: it can chat and provide PQ values for a specific user, as well as analyze a defect and rank it according to the PQ and Craftsmanship score values. The model is not very accurate when it comes to image analysis, since the provided data contain a very small amount of samples, around ten images per category. If more data were available, the model should be even more accurate and general, which means that it would not be overfitted. Furthermore, another improvement would be to create a customized LLM. The model now is trained on photos of just two models, in future releases it should be good if the dataset also includes other car models to get a wider range of errors.

In terms of latency, the system shows varying response times across different components. Image analysis latency ranges from approximately 648 ms to 889 ms, which is acceptable for near-real-time processing. Vector database queries without LLM integration are significantly faster, averaging around 92 ms, while queries routed through the GPT-4.1 API introduce additional overhead, with response times between 2.248 seconds and 3.18 seconds. This latency profile suggests that while the core image processing is efficient, the interaction with the language model API remains the primary bottleneck. Optimizing the integration between the vector database and the LLM or considering alternative architectures could improve overall system responsiveness.

# Bibliography


[1] K. Stylidis, C. Wickman, and R. Söderberg, "Perceived quality in the automotive industry: A comprehensive framework," *SAE International Journal of Materials and Manufacturing*, vol. 7, no. 3, pp. 731–738, 2014, accessed February 21, 2025.

[2] K. Stylidis, "Perceived quality of cars: A novel framework and evaluation methodology," Ph.D. dissertation, Chalmers University of Technology, Gothenburg, Sweden, 2020.

[3] V. Zolfaghari, N. Petrovic, F. Pan, K. Lebioda, and A. Knoll, "Adopting RAG for LLM-aided future vehicle design," in *Proceedings of the IEEE FLLM 2024*, 2024.

[4] Databricks, "What are vector databases? definition and uses," Databricks Glossary, accessed March 24, 2025. [Online]. Available: https://www.databricks.com/glossary/vector-database

[5] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, Nov 1975, accessed June 24, 2025.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006, accessed June 24, 2025.

[7] M. Cheatsheet, "Concepts — ML glossary documentation," https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html, accessed March 19, 2025.

[8] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016, accessed March 2, 2025. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf

[9] A. Vidhya, "Guide to prevent overfitting in neural networks," 2021, accessed February 28, 2025.

[10] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[11] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Neural Computation*, vol. 19, no. 10, pp. 2501–2520, 2007, accessed March 7, 2025.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, accessed March 12, 2025.

[13] S. Minaee, A. A. Kazerouni, M. A. Khandwala, M. Gholami, A. Gupta, Q. Yang *et al.*, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, Feb 2024, accessed March 21, 2025.

[14] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in large language models: a comprehensive review," *arXiv preprint arXiv:2310.14735*, 2023, accessed March 26, 2025.

[15] J. vom Brocke, A. Simons, S. Riemer, P. Schieder, R. C. Thomas, and M. M. L. H. A. Schieder, "Application programming interface (api) research: A review of the past to inform the future," *Communications of the ACM*, vol. 62, no. 6, pp. 26–35, 2019, accessed February 14, 2025.

[16] M. Lewis, E. Perez, A. Piktus, and V. Y. Yamada, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 4578–4587, 2021, accessed April 1, 2025.

[17] A. Bechard, M. Kassner, C. Liu, D. L. Lee, and A. Phung, "Reducing hallucination in structured outputs via retrieval-augmented generation," *arXiv preprint arXiv:2402.03561*, Feb 2024, accessed March 9, 2025.

[18] N. Reimers and I. Gurevych, "Making monolingual sentence embeddings multilingual using knowledge distillation," 2020, accessed: 2025-06-25.

[19] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *arXiv preprint arXiv:1702.08734*, 2017.

[20] M. Abadi, P. Barham, J. Chen *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] E. Y. Boateng and D. A. Abaye, "A review of the logistic regression model with emphasis on medical research," *Journal of Data Analysis and Information Processing*, vol. 7, no. 4, 2019, accessed June 20, 2025.

[23] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 114, 2016, accessed June 20, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271616000265

[24] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," *arXiv preprint arXiv:1706.09516*, 2017, accessed June 20, 2025. [Online]. Available: https://arxiv.org/pdf/1706.09516.pdf