LLM-Based Data Extraction and Machine Learning for CO2e Estimation of Semiconductor Components

ARVID MÜLLER & ELIAS FLYNN ROSENBERG
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

LLM-Based Data Extraction and Machine Learning for CO_2e Estimation of Semiconductor Components

Arvid Müller ar3557mu-s@student.lu.se
Elias Flynn Rosenberg el5374fl-s@student.lu.se

Department of Electrical and Information Technology

Lund University

in collaboration with

Volvo Cars

Supervisor: Amir Aminifar

Assistant Supervisor: Baichuan Huang

Supervisor (Volvo Cars): Golnaz Afshar

Examiner: Christian Nyberg

June 14, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

This thesis investigates the application of Artificial Intelligence (AI) and Machine Learning (ML) methods to improve carbon footprint estimation for Integrated Circuit (IC) components in automotive Cost Engineering (CE). The work addresses challenges faced by electrical cost engineers at Volvo Cars: the manual, time-consuming process of collecting component specifications, and estimating Carbon Dioxide Equivalent (CO_2e) emissions from electronic components.

We developed an automated data collection pipeline using Large Language Models (LLMs) to extract structured information from manufacturer datasheets and material content sheet documents. Three models, Gemma 3 4B, Llama 3.1 8B, and gpt-oss 120B, were evaluated for extraction accuracy, inference time and memory usage. The gpt-oss 120B model achieved 98.5% extraction accuracy for the validation set. The data extraction pipeline was then applied to a larger set of datasheets and material content sheets and converted unstructured PDF documents into structured tabular data to be used for machine learning.

We developed machine learning models to predict the masses of four hotspot metals, copper, gold, palladium and silver, which are the primary contributors to raw-material carbon emissions. We evaluated TabPFN (a transformer-based prior-fitted network), XGBoost and CatBoost regression models. CatBoost achieved the best overall performance, with R^2 values of 0.994 for gold and 0.971 for palladium. SHAP analysis revealed that total component mass is the most important feature for copper content, while pin count has the biggest effect on gold and silver predictions. We introduced a simplified hotspot scaling approach that reduces the previous model's complexity from ten function-type-specific scaling factors to a single global factor, with minimal loss in accuracy.

The model was integrated into a desktop application that enables cost engineers to retrieve component specifications, predict metal content when material data are unavailable, and compute CO_2e emissions automatically. The tool reduces manual effort and provides a reproducible, consistent framework for carbon footprint estimation in component evaluation.

This work demonstrates that AI and ML methods can effectively automate and improve semiconductor carbon footprint estimation, supporting Volvo Cars' decarbonisation objectives while reducing the manual workload of cost engineering teams.

Popular Science Summary

Modern cars contain hundreds of electronic components, tiny chips that control everything from the engine to the entertainment system. But these small components come with an environmental cost, the carbon dioxide released during their production. For a company like Volvo Cars, which aims to eliminate all greenhouse gas emissions by 2040, understanding and reducing the carbon footprint of every single component is crucial.

The problem is that tracking these emissions is incredibly time-consuming. Engineers must manually search through technical documents from manufacturers, extract information about each component's size, weight and material composition, and then calculate the associated carbon emissions.

This thesis tackles that challenge by using machine learning to make that process easier. We developed a system that uses artificial intelligence to automatically read and extract information from manufacturer documentation. The AI can scan through technical PDFs, find the relevant information (such as how much gold, silver, or copper a component contains), and organize it into a structured format.

However, manufacturers don't always publish complete material information, so we also developed machine learning models that can predict a component's metal content based on characteristics like its size, weight and number of pins. The models proved accurate, particularly for gold, the metal that contributes most to carbon emissions from raw-materials in electrical components.

All of this technology was packaged into a user-friendly desktop application that engineers at Volvo Cars can use in their daily work. By entering a component's identification number, the tool can retrieve specifications, predict metal content if needed, and calculate the estimated carbon footprint. What once took an excessive amount of time and manual labor, now takes seconds, and what once required expert knowledge is now accessible to any engineer.

Statement of Originality

We hereby affirm that generative AI was used during the creation of this thesis. AI-tools were used to revise spelling, phrasing and grammar in order to ensure a consistent and high-quality language usage. AI was also used in coding contexts to help with syntax and algorithm suggestions. Everything was reviewed by us, and we take full responsibility for the content. All ideas, analysis and conclusions are our own.

Table of Contents

List of Figures

List of Tables

Abbreviations

AI — Artificial Intelligence

ML — Machine Learning

NLP — Natural Language Processing

LLM — Large Language Models

LCA — Life Cycle Assessment

ECU — Electronic Control Unit

IC — Integrated Circuit

API — Application Programming Interface

CE — Cost Engineering

PDF — Portable Document Format

MCS — Material Content Sheet

CART — Classification and Regression Trees

MPN — Manufacturer Part Number

GUI — Graphical User Interface

CO_2e — Carbon Dioxide Equivalent

Chapter 1
Introduction

Recent advancements in artificial intelligence (AI) and machine learning (ML) have led to their widespread adoption across a variety of industries, where they are increasingly used to improve efficiency, automate complex tasks, and support data-driven decision-making. In engineering-intensive domains, ML has demonstrated particular value in extracting patterns from large, heterogeneous datasets that are difficult to analyse using traditional methods alone.

Cost engineering is one such domain that stands to benefit significantly from these advancements. Cost engineers are responsible for cost-optimised design solutions and competitive costs through actively providing fact-based inputs and negotiation support to reach optimal prices. Cost engineers are required to estimate, analyse, and manage product costs throughout the development lifecycle, often under conditions of uncertainty, incomplete information, and time pressure. In parallel, sustainability considerations, such as carbon footprint estimation, are becoming integral to product development, further increasing the complexity of cost-related analyses.

This thesis explores how AI and ML techniques can be applied to carbon footprint estimation for electronic components, with the aim of supporting and enhancing the work of cost engineers rather than replacing traditional methods. By using LLMs to collect and extract large amounts of manufacturer data, the thesis then investigates whether ML-based methods can enhance the currently used model for carbon footprint estimation.

1.1 Background

The following description of the workflow and challenges is based on observations and discussions conducted during the project at Volvo Cars, unless otherwise stated.

At Volvo Car's Electrical Cost Engineering department, we were met with a group of highly specialised engineers, working with analysing and estimating the cost and carbon footprint of electronic components, ranging from simple passive components to complex Electronic Control Units (ECUs). The supply chain of ECUs is convoluted and consists of multiple different manufacturers and suppliers, meaning that Cost Engineers need to achieve an optimised design solution and competitive cost through actively providing fact-based input and negotiation

support to reach the best price by setting an affordable target price and providing cost calculations based on state-of-the-art material and process technology. The cost engineering department delivers and explains fact-based cost estimates, reviews and analyses quotations, supports negotiations, and works in cross-functional teams.

As climate change continues, Volvo Cars, as a mobility provider, recognises itself as a contributor and therefore their responsibility to act accordingly. Volvo Cars aims to reach net-zero greenhouse gas emissions by 2040 by reducing emissions across its entire value chain [1]. As a result, cost engineers, in addition to cost, need to study, estimate and conduct Life Cycle Assessments (LCAs) for carbon emissions and minimise environmental impact throughout all purchased components used in vehicles, ranging from small electronic components to larger metal or plastic parts. This is achieved using a combination of experience, market knowledge, qualified judgment and reverse-engineering of components' composition and origin. This meant that the workflow involved a high degree of manual effort, motivating interest in increased automation. They therefore expressed an interest in using AI/ML to aid their work.

A previous thesis project by Zhang et al. [2] resulted in a model for semiconductor carbon footprint estimation based on raw material content and semiconductor process for electronic components. This model was based on linear regression with component physical parameters as independent variables and metal content weights as dependent variables. The model assumed linearity between features and dependent variables. We identified this linear dependency as an area of improvement, where, instead of linear regression, more sophisticated ML techniques could be used to capture the non-linearities and potentially improve the accuracy of the metal mass predictions, and in turn, carbon emissions estimation.

Training these ML models would require a large amount of data. At the start of the project, however, there was no readily available or sufficiently structured database of electronic component parameters suitable for this purpose, and only limited documented records where such parameters had been identified. This meant that data would have to be collected from manufacturers, most of which could be found in data sheets and material content sheets on the manufacturer's webpage. This information would have to be efficiently and reliably extracted. A large part of the work would thus be to create a way to build a dataset, but also a way for the electronic cost engineers to efficiently get component parameters, without having to find them manually. Admittedly, an accurate metal content weight prediction is only part of a larger model containing additional variables. Nevertheless, the metal content weight estimation was the aspect of the existing model deemed improvable, given the limited resources and duration of the project.

1.2 Objective

The objectives of the project were as follows:

- Set up an automated pipeline to collect and extract electronic component specifications from manufacturer-provided data sheets and material content sheets using Natural Language Processing (NLP) and LLMs.

- Use the collected data to improve the accuracy of the current material mass estimation model using ML, which in turn could lead to more reliable CO_2e emissions estimations.
- Integrate the data collection pipeline in the workflow of electrical cost engineers to help reduce manual efforts.

1.3 Boundaries

In order to keep the scope and scale of the project at a manageable size, the core focus was on semiconductor IC components. Furthermore, the availability of data on certain components was restricted, meaning that only what was publicly available via manufacturers' websites and third-party providers could be used.

Preferably, the LLM pipeline should be runnable locally on a common office laptop, which also puts restrictions on the size of the LLMs to be used. The LLMs in question also had to be open source in order to allow free use.

1.4 Contributions

The construction of the dataset was a large part of this thesis and is based on methods for using LLMs to extract structured data from unstructured sources, a topic which has gotten attention in research lately [3, 4, 5].

The thesis also explores different ML implementations for predicting metal content in IC components, given a certain set of input features. Approaches tested were different gradient boosted models as well as TabPFN [6, 7]. Moreover, SHAP was employed to try to explain the link between these input features and the presence of different metals.

Additionally, the work aims to aid Electrical Cost Engineers at an industry-leading company in reaching decarbonisation goals, continuing the work of Zhang et al. [2] and making their workflow more general and efficient.

The project resulted in an app with a user-friendly Graphical User Interface (GUI) to aid the cost engineers by lowering manual labour and calculations and providing estimated CO_2e emissions for semiconductor components.

Chapter 2
Related Work

2.1 LCA of Semiconductors

A Life Cycle Assessment (LCA) is a methodology for quantifying the environmental impacts of a product across its entire lifespan, from raw material extraction through manufacturing, usage, and disposal. In the context of ICs, LCA has received growing attention as the industry faces increasing pressure to reduce its carbon footprint. However, performing a rigorous LCA for electronic components presents several fundamental challenges. Manufacturing processes, complex global supply chains, and a lack of standardised data collection make it difficult to obtain the life cycle inventory data needed for accurate assessments. Life cycle inventory databases available today lack the data needed for advanced technology nodes, and what is missing is a higher granularity of results at the process and fab level [8]. Similarly, DeltaLCA [9] highlights that performing a rigorous LCA requires significant expertise and can take months for complex designs, making it impractical to compute assessments for multiple design variations during the design process itself.

At semiconductor packaging level, Kuo et al. [10] conducted an LCA comparing different IC packaging technologies. Their results show that packaging choices influence environmental impact, with gold wire identified as a large contributor to environmental damage across all package types. Their work shows how the choice of package type affects material composition, which connects directly to the use of package class as an input feature in the models developed throughout this thesis.

2.2 Previous Thesis Project at Electrical Cost Engineering

Previous work related to the LCA of components had been done at Volvo's Electrical Cost Engineering department by Zhang et al. [2]. The research was conducted with the goal of making carbon footprint estimation for semiconductors transparent. The thesis had three subgoals. First of all, to develop a raw material carbon emission model. This involved collecting carbon emission data related to raw materials and estimating the amount of raw materials for individual components. The second goal was to create a manufacturing carbon emission model. The third and final goal was to integrate and validate the combination of both models as a final carbon footprint model. The idea was to combine the raw material model with the semiconductor

manufacturing model, and then validate it against electronic component supplier emission numbers.

Focusing on the raw material carbon emission model, the emission estimation model by Zhang et al. [2] considers only the part of the LCA that concerns environmental impact during the manufacturing phase, including the collection of materials all the way to the finished product. This model does not include the product's use phase or disposal. This means that the model is simplified for industrial usability.

Zhang et al. [2] define a carbon emission metal hotspot map as a method for identifying key carbon emission drivers. Using this method, the paper identifies copper, gold, palladium, and silver as the main contributors.

Zhang et al. [2] use linear regression models to estimate hotspot metal masses and then predict CO_2e emissions per component. They identify the main raw-material carbon-emission hotspots as Au, Pd, Ag, SiO_2, and Cu, which account for 85.91%, 4.65%, 6.38%, 1.17%, and 1.17% of raw-material carbon emissions, respectively, with the remaining materials accounting for 0.71%. The paper splits the analysis based on whether the components contain gold or not, which changes the hotspot dominance to a large extent. Table 4.4 in [2] presents a hotspot map for components that contain gold, and Table 4.5 in [2] includes components that do not contain gold.

In Zhang et al., semiconductor components are grouped into five categories: power management ICs, logic ICs, interface ICs, microcontrollers, and semiconductor memories. The regression models use component-level features such as length, height, mass, pin count, and package type. These parameters were also chosen for this thesis due to their high recurrence and availability from public sources.

Zhang et al. [2] fit their linear regression models using the least-squares method. For the power management IC regressions in Table 4.6 , they report coefficients of determination above 0.8 and therefore conclude that the models perform within a reasonable range. However, one reported value, $R^2 = 85.0438$, is inconsistent with the conventional interpretation of R^2 and likely reflects either a reporting error or a different scaling. Together with the broader limitation of assuming linear relationships between features and metal masses, this motivates our decision to revisit the modelling step using more flexible ML methods. Appendix B in [2] is rewritten as Table 2.1.

For a new component, Zhang et al. [2] first estimate the masses of hotspot metals such as Au, Ag, Pd, or Cu using the regression equations. These predicted metal masses are then multiplied by the corresponding carbon emission factors to obtain hotspot-metal emissions. Finally, the summed hotspot emissions are scaled by the hotspot share reported to estimate total raw-material CO_2e emissions per component.

Another data point that is argued for is the die-to-package ratio, i.e, how much of the component package is filled by silicon dies. This is used to calculate carbon emissions at a wafer level. The exact ratios per component are not provided by manufacturers, thus Zhang et al. [2] use the approximated values shown in Table 2.3 (Table 4.7 in Zhang et al. [2]). The five package types (sometimes referred to as package classes) considered can be seen in Table 2.2. There exist many variations of these packages for different manufacturers and components, but ICs can in general

be classified into one of these five classes.

Component category	**Metal**	R^2	**MSE**
Interface IC with gold	Au	0.8234	0.00066
Interface IC with gold	Ag	0.953	0.00289
Interface IC with gold	Pd	0.6147	0.00000805
Interface IC without gold	Ag	0.953	0.00289
Interface IC without gold	Cu	85.0438	0.9815
Logic IC with gold	Au	0.904	0.000000476
Logic IC with gold	Ag	0.746	0.0000926
Logic IC with gold	Third hotspot term	N/A	N/A
MCU with gold	Au	0.999	1.54×10^{-30}
MCU with gold	Ag	0.818	2.540
MCU with gold	Third hotspot term	N/A	N/A
Logic IC without gold	Ag	0.7464	0.0000926
Logic IC without gold	Cu	0.996	0.5695
MCU without gold	Ag	0.818	2.540
MCU without gold	Cu	0.999	8.720

Table 2.1: Regression quality reported in Appendix B of [2].

Package class
DIP — Dual In-line Package
SOP — Small Outline Package
QFP — Quad Flat Package
BGA — Ball Grid Array
CSP — Chip Scale Package

Table 2.2: Package classes.

Package type	**Die-to-package ratio**
DIP	10%–30%
SOP	20%–40%
QFP	30%–50%
BGA	50%–70%
CSP	80%–95%

Table 2.3: Estimated die-to-package ratio ranges by package type (die area divided by package area).

Chapter 3
Theory

3.1 Cost Engineering

Cost engineering is a discipline that spans planning, bottom-up estimation, control, and optimisation of costs throughout the lifecycle of a project [11]. Methods from fields such as engineering, economics, and project management are used to support decision-making by understanding the financial impacts and resources used. Normally, cost engineering focuses on both direct and indirect costs such as upfront costs and others like materials, labour, equipment, and overhead. Cost engineering also has to take risk into account while trying to achieve cost efficiency, meet budget constraints, and satisfy the requirements.

In recent years, cost engineering has also started to include environmental considerations, especially greenhouse gas emissions, as organisations increasingly recognise the importance of climate impacts and decarbonisation policies. In this context, carbon footprint is treated as a key decision parameter, steered in parallel with cost, throughout the lifecycle of a product or project. By integrating carbon footprint into cost engineering frameworks, it becomes possible to make more complete evaluations of material and supplier alternatives by considering not only immediate costs, but also future regulations and sustainability goals.

Overall, cost engineering with a focus on carbon footprint and component prices provides a structured framework for understanding how to minimise both production costs and carbon emissions.

3.2 Machine Learning

Machine learning (ML) is a field of artificial intelligence that focuses on developing algorithms and statistical models that enable computer systems to learn patterns from data and improve their performance on specific tasks without being explicitly programmed. By leveraging techniques such as supervised learning, unsupervised learning, and reinforcement learning, other machine learning methods can identify complex relationships, make predictions, and support decision-making across a wide range of applications. Central to machine learning is the concept of generalisation, whereby a model trained on historical data can effectively handle unseen data. The field draws on principles from statistics, optimisation, and computer science, and

has become a foundational approach for analysing data to make predictions based on large and complex datasets in both research and industry [12].

3.3 Baseline Model: Linear Regression

Linear regression is a supervised learning method trained on labelled data. It models a dependent variable y from independent variables x. The model does this by fitting a linear relationship that best matches the training data. In Zhang et al. [2], the dependent variables are the metal contents, and the independent variables are features of the components.

In general, the model can be written as

$$y^{(i)} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j + \varepsilon, \tag{3.1}$$

where $y^{(i)}$ is the content of target metal i (for example in mg for Ag, Au, Cu, and Pd), x_j are the independent variables (component features), β_j are the coefficients learned by the model, p is the number of input features, and ε is the error term.

Linear regression learns the coefficients β by fitting the model to data. In Zhang et al. [2], this is done using least squares, which finds the coefficients that minimise the sum of squared residuals:

$$\hat{\beta} = \arg\min_{\beta} \sum_{n=1}^{N} \left(y_n^{(i)} - \hat{y}_n^{(i)} \right)^2, \tag{3.2}$$

$$\hat{y}_n^{(i)} = \beta_0 + \sum_{j=1}^{p} \beta_j x_{n,j}, \tag{3.3}$$

where $\hat{\beta}$ are the estimated coefficients, N is the number of observations in the dataset, p is the number of input features, $y_n^{(i)}$ is the true value of target metal i for observation n, and $\hat{y}_n^{(i)}$ is the corresponding predicted value. This assumes a linear relationship between features and metal mass.

3.3.1 Model Evaluation

Zhang et al. [2] report model performance using mean squared error (MSE) and coefficient of determination (R^2).

Mean squared error measures the average squared difference between predictions and observed values:

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} \left(y_n^{(i)} - \hat{y}_n^{(i)} \right)^2. \tag{3.4}$$

Here, N denotes the number of observations used in the evaluation. MSE is simply how far the fitted line is from the data points on average, but with squared errors to penalise larger deviations. Zhang et al. [2] does not clearly describe the evaluation on unseen data. If MSE is computed on the same data used for fitting, it mainly reflects in-sample fit rather than generalisation.

The coefficient of determination R^2 measures how much of the variance in $y^{(k)}$ is explained by the model:

$$R^2 = 1 - \frac{\sum_{n=1}^{N} \left(y_n^{(i)} - \hat{y}_n^{(i)} \right)^2}{\sum_{n=1}^{N} \left(y_n^{(i)} - \bar{y}^{(i)} \right)^2}, \tag{3.5}$$

$$\bar{y}^{(i)} = \frac{1}{N} \sum_{n=1}^{N} y_n^{(i)}. \tag{3.6}$$

Again, N is the number of observations, and $\bar{y}^{(i)}$ is the sample mean of the target metal i. The score lies in the interval $(-\infty, 1]$, where values closer to 1 indicate that the features explain the dependent variable well, and values nearing $-\infty$ indicate weak explanatory power. A score below 0 means that the model is worse at prediction than just using the average.

The regression outputs fitted equations together with R^2 and MSE. Depending on the regression setting, Zhang et al. [2] distinguishes between components with gold and without gold. With gold, the hotspot metals are Au, Ag, and Pd, while without gold, the hotspot metals are Ag and Cu. Zhang et al. [2] reports that all regression equations for power IC components have $R^2 > 0.8$ and a reasonable MSE.

A limitation is that the model lacks a clear train-test split to assess performance on unseen components. The evaluation mainly relies on reported R^2 and MSE, which makes it difficult to judge how well the model generalises beyond the fitted dataset. The linear regression model is trained on over 200 components, and the classification of function type is done manually. There is no data on the exact number of samples per function type, or for components with or without gold.

3.4 Interquartile Range (IQR) Outlier Detection

The interquartile range (IQR) is a robust measure of statistical dispersion, defined as the difference between the 75th percentile (Q_3) and the 25th percentile (Q_1) of a distribution:

$$\text{IQR} = Q_3 - Q_1. \tag{3.7}$$

Unlike the standard deviation, the IQR is resistant to extreme values because it is computed only from the middle 50% of the observations. A standard deviation based outlier detection method, where outliers are flagged if they fall outside $\mu \pm k\sigma$, would be strongly affected by extreme values, since such values influence both the mean and the standard deviation. This can shift the thresholds substantially when the outlier is very large or very small. The IQR method is therefore more robust. This makes IQR well-suited to detect outliers in datasets where occasional extraction or unit-conversion errors can produce unrealistic numerical values.

Outlier boundaries can be defined using the IQR method [13]:

$$L = Q_1 - k \cdot \text{IQR}, \qquad U = Q_3 + k \cdot \text{IQR}, \tag{3.8}$$

where L and U denote the lower and upper fences, respectively, and k is a sensitivity parameter. In this thesis, $k = 1$ was used for the screening of within-package-type groups. The motivation for $k = 1$ was that components belonging to the same package type are expected to cluster tightly around a nominal physical size, so even moderate deviations are more likely to reflect data extraction errors, rather than genuine physical variation.

3.5 Large Language Models

For the purpose of this thesis, language models based on the transformer architecture introduced in *Attention Is All You Need* [14] are used. The main reason this architecture is relevant is that self-attention lets each token incorporate information from other parts of the sequence. This matters for manufacturer documents, where features and values are spread across sections, tables, and where correct interpretation depends on context.

3.5.1 Transformer Architecture

Transformer models operate on tokenised text. A tokeniser converts raw text into a sequence of tokens, and each token is mapped to a vector representation (an embedding). Because the model has no recurrence or convolution, positional encodings are added so that word order is represented.

The model is built as a stack of layers. Each layer contains a multi-head self-attention mechanism and a position-wise feed-forward network. Residual connections and layer normalisation are applied around each sublayer. Self-attention updates each token by combining information from other tokens. Learned query, key, and value projections are used to compute attention weights, which then form a weighted sum of value vectors. Multi-head attention runs this process in parallel with different projections and concatenates the outputs. The feed-forward sublayer applies two linear transformations with a non-linearity in between, independently at each position.

The original Transformer is an encoder-decoder. The encoder maps the input sequence to continuous representations. The decoder generates the output autoregressively, with masking so that each position depends only on previous positions and on the encoder output through encoder-decoder attention. In inference, decoding is typically done with beam search and a length penalty, which affects the stability of generated sequences. Before a token is sampled, the model produces a probability distribution over the vocabulary via a softmax over the final logits [14].

Temperature scaling controls the sharpness of this distribution by dividing the logits by a scalar value before the softmax is applied. A temperature below one sharpens the distribution, making the model more likely to select high-probability tokens and producing more deterministic outputs. A temperature above one flattens the distribution, increasing the likelihood of lower-probability tokens and producing more varied or creative output. Temperature is therefore a simple but effective way to trade off between output diversity and output reliability at inference time [15].

3.5.2 Memory Requirements of LLMs

Running inference with an LLM requires holding two different categories of data in memory simultaneously. The first is the **model weights**, which are proportional to the number of parameters of a model, and the numerical precision used to store them. The required memory for the weights can be approximated as:

$$Mem_{weights} = N_{params} \times B_{precision},$$

where N_{params} is the number of parameters and $B_{precision}$ is the number of bytes per parameter (e.g., 2 bytes for FP16/BF16 and 4 bytes for FP32). Consequently, a model with 4 billion parameters, stored in BF16, requires approximately 8 GB of memory just for the weights [16].

The second major component is the **key-value (KV) cache**. During autoregressive generation, transformer models reuse the key and value representations of all previously processed tokens to avoid redundant computations. The KV cache size per token is given by:

$$\text{KV cache per token (bytes)} = 2 \times N_{layers} \times (N_{heads} \times d_{head}) \times B_{precision},$$

where the factor 2 is because of both K and V matrices, $N_{head} \times d_{head}$ commonly equals the model's *hidden size* or *dimension* [16].

Together, these two components define the **peak memory** required during inference. Peak memory usage during generation is approximately equal to the sum of the weight memory and the KV cache memory [17].

3.5.3 Quantization

Given the large memory requirements of modern LLMs, quantization is being used as a technique for enabling model deployment on hardware with limited memory capacity. Quantization significantly reduces a model's required memory by lowering the numerical precision of a model's parameters, while avoiding performance loss [18].

Quantization works by replacing high-precision float representations (e.g., FP32 and BF16) with lower-bit formats. Common formats include FP16, INT8 and INT4 [19].

3.5.4 Relevance to Document Extraction

Three properties of the transformer architecture are directly relevant to our use case. First, self-attention provides a mechanism for linking features and values across the document, which is important when information is distributed over many pages or sections. Second, self-attention has quadratic per-layer computational complexity that grows with sequence length, so the model operates on finite-length token sequences. This limits how much of a PDF can be processed in one forward pass. Third, because output is generated sequentially, the decoding strategy affects consistency and error propagation when we require structured output (for example,

JSON conforming to a schema). We rely on these properties when using a pre-trained language model to extract structured fields from datasheets and material content sheets.

3.5.5 Unstructured-to-Structured Data

A prerequisite for supervised learning is structured data collection. This means that each component must conform to a predefined schema, where the same set of features and target variables is stored for every observation using consistent data types, such as strings, integers, and floats. This, however, requires the extraction system to return outputs that map to our supervised learning input requirement. In addition, it requires the returned information to be reproducible, auditable, stable, and accurate.

3.5.6 Schema as an Interface Contract

To achieve reliable and reproducible results, a schema for interfacing with the LLM is important. Using a consistent schema allows for easier API interactions, making inputs and outputs structured to and from the LLM. The JSON schema was chosen because it specifies expected structure, data types and enables automatic validation for LLM outputs. This allows validation to detect errors, and enables the system to reject non-compliant responses and optionally retry the request. Additionally, JSON makes accessing LLMs through different APIs easier due to high compatibility. [20]

Using JSON for the LLM response allows for setting clear rules for output via schema-based parsing and validation. The pipeline interprets the LLM response and validates it against the schema. The expected structure can be structured using a model from Pydantic. This JSON Schema can be derived from the Pydantic model and used as communication to the LLM. This provides a structure for the LLM to model. It therefore has an explicit structure to use. Validation can then happen in two steps: checking that the return format is JSON, then Pydantic checks for the correct types and constraints. After the return has been validated, the JSON format allows conversion to other representations and to tabular format, to be used for supervised learning [20, 21].

3.6 Manufacturer Documentation and Data Characteristics

3.6.1 Overview and Data Availability

The two main sources of data for electronic components are manufacturer datasheets and material content sheets (MCS). These are commonly provided by manufacturers on their websites, although some manufacturers do not make this information public. Often, the content is provided as PDFs, which means the data is either unstructured or presented in natural language. Sometimes the datasheets or material content sheets have missing fields, or important features cannot be derived because the information is not explicit. In other cases, the datasheet or MCS do not exist, or is not publicly provided. As a result, purely structured data for components is less

common. This means that information extraction might be challenging and could reduce efficiency. [22]

3.6.2 Datasheets

Electronic component datasheets contain features, applications and technical specifications of components. This is relevant when deciding whether a component is fit for an intended purpose, since it describes capabilities and limitations. Reading datasheets is standard practice for understanding a component and its characteristics. These features define the component and help classify its usage and purpose. The most relevant features for our application are function type and package dimensions.

3.6.3 Material Content Sheets

Material content sheets contain the product's sales number, also known as the Manufacturer Part Number (MPN), as well as package type, total weight, and the substances the component contains. This ranges from silicon to chemicals to pure metals, including the mass of each substance in the component. The most relevant fields are manufacturer, manufacturer part number, and substance mass in mg. Material content sheets are also a practical way of understanding what a component can deliver and what applications are feasible. The material composition of a component is vital for building a structured LCA, since different metals contribute very differently to CO_2e emissions.

These sheets form the basis for building a structured dataset for modelling and are used to support the LCA process.

3.7 Ensemble Learning

Ensemble learning is a class of ML methods that aims to aggregate outputs from multiple small, weak learners in order to improve predictive performance [23].

Broadly, ensemble methods are divided into, among others:

- **Bagging methods**, which primarily reduce variance by training models independently on resampled data [23].
- **Boosting methods**, which build models iteratively, with each new learner focusing on correcting errors made by previous learners [23].

3.7.1 Gradient Boosting

Gradient Boosting is an ensemble learning technique which iteratively builds a strong predictive model by combining multiple weak learners, often Classification and Regression Trees (CARTs). It aims to minimise errors by making each new learner approximate the negative gradient of a specified loss function.

Let $L(y, \hat{y}(x))$ denote a differentiable loss function, where y is the true label of a data sample and $\hat{y}(x)$ is the current ensemble prediction for the same sample. The objective is to minimise the total loss over the training dataset:

$$\mathcal{L} = \sum_{i=1}^{n} L(y_i, \hat{y}(x_i)). \tag{3.9}$$

At iteration s, the ensemble predictor is updated as:

$$\hat{y}_s(x) = \hat{y}_{s-1}(x) + \eta h_s(x), \tag{3.10}$$

where $\hat{y}_s(x)$ is the ensemble prediction after boosting stage s, $\hat{y}_{s-1}(x)$ is the ensemble prediction from the previous stage, $h_s(x)$ is the base learner fitted at stage s, and $\eta \in (0, 1]$ is the learning rate.

The idea is that $h_s(x)$ is trained to approximate the negative gradient of the loss function, evaluated at the current model:

$$r_{is} = -\left[\frac{\partial L(y_i, \hat{y}(x_i))}{\partial \hat{y}(x_i)}\right]_{\hat{y}=\hat{y}_{s-1}} = -\frac{\partial L(y_i, \hat{y}_{s-1}(x_i))}{\partial \hat{y}_{s-1}(x_i)} \tag{3.11}$$

Thus, the algorithm performs gradient descent in function space. As mentioned, CARTs are often used as weak learners due to their flexibility and ability to model nonlinear interactions [24].

XGBoost

XGBoost (Extreme Gradient Boosting), introduced in [25], is an optimised and regularised implementation of gradient boosting designed for computational efficiency and predictive performance.

Its objective function consists of a training loss term and a regularisation term:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k), \tag{3.12}$$

where t denotes the current boosting iteration, f_k represents a regression tree added at iteration k, and the regularisation term penalises model complexity:

$$\Omega(f) = \gamma N_{leaves} + \frac{1}{2}\lambda ||w||^2, \tag{3.13}$$

where N_{leaves} is the number of leaves in the tree, w are leaf weights, γ is the leaf penalty, and λ is an L2 regularisation parameter.

XGBoost differs from regular gradient boosting in several ways. It uses second-order optimisation, using both first- and second-order derivatives (Hessian) to approximate the loss function via second-order Taylor expansion. It also uses regularisation to reduce overfitting. Moreover, it optimises the process by featuring parallel tree construction, cache optimisation and distributed computing support, making it scalable. These enhancements often lead to superior performance in tabular datasets [25].

CatBoost

CatBoost (Categorical Boosting)is a gradient boosting algorithm specifically designed to handle categorical features effectively and reduce overfitting.

Introduced by Yandex in [26], it is distinguished by two central innovations, ordered boosting and native handling of categorical variables.

In regular gradient boosting, overfitting may occur due to target leakage when residuals are computed using the same data used to train the model. CatBoost aims to avoid this by using **ordered boosting**, where target statistics and residuals are computed using permutations of the dataset that simulate a real learning process. This reduces bias and improves generalisation.

CatBoost also avoids relying on one-hot encoding or manual preprocessing by transforming categorical features into numerical representations using target-based statistics, computed in an ordered manner to prevent leakage. This reduces dimensionality compared to one-hot encoding, improves efficiency, and preserves predictive signal in high-cardinality categorical variables [26].

3.8 TabPFN

Tabular Prior-Fitted Network (TabPFN) is a foundation model designed for supervised learning on tabular datasets, in particular, datasets of smaller size. Introduced by Hollman et al. [6], it is based on a Transformer architecture, pre-trained on a larger distribution of synthetic datasets in tabular form. During inference, TabPFN performs prediction via in-context learning; the training data is provided as part of the model's input, allowing it to approximate Bayesian inference without task-specific training or hyperparameter tuning. This enables fast, one-shot or few-shot predictions that often achieve competitive performance with traditional methods such as gradient boosting while requiring less computational overhead. Since the model is pre-trained, TabPFN can handle tabular problems well, even with limited data or when fast results are required.

3.8.1 Bayesian Formulation

TabPFN can be seen as a model that approximates Bayesian inference for supervised learning problems. Instead of training a separate model for each dataset, it is trained in advance to directly approximate the Bayesian posterior predictive distribution across many possible tabular tasks.

Consider a labelled dataset:

$$D = \{(x_i, y_i)\}_{i=1}^{n},$$

where $x_i \in \mathbb{R}^d$ are feature vectors and y_i are target labels.

In Bayesian learning, a model is assumed as $p(y|x, \theta)$ with parameters θ and a prior distribution $p(\theta)$ is placed over these parameters. The observation of dataset D leads to updating the belief of the parameters using Bayes theorem:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}. \tag{3.14}$$

In order to make predictions for a new input sample x_*, the posterior predictive distribution is calculated:

$$p(y_*|x_*, D) = \int p(y_*|x_*, \theta)p(\theta|D)d\theta. \tag{3.15}$$

This means that predictions are averaged over all possible parameter values, weighted by how likely they are given the data. In practice, this integral is very difficult or even impossible to compute exactly.

TabPFN circumvents this problem by avoiding calculating the posterior over parameters and instead learning a predictive mapping:

$$q_\phi(D, x_*) \approx p(y_*|x_*, D), \tag{3.16}$$

where q_ϕ denotes the predictive mapping learned by the network, and ϕ are the pretrained Transformer parameters. During inference, the model does not update its weights. Instead, it takes the full training dataset D and the new sample input x_* and directly outputs a predictive distribution. In this manner, TabPFN learns the entire Bayesian inference in one single forward pass. This process is known as amortised inference. The computational effort required to approximate the Bayesian inference is moved to the pretraining phase, resulting in inference on new data being very fast.

3.8.2 Synthetic Task Prior

The synthetic task prior is a very important part of TabPFN. The prior determines what kind of tabular prediction problems the model expects to encounter. Unlike classical Bayesian models, where the priors are defined directly over the parameters, TabPFN defines its prior through the distribution of datasets used during pretraining.

Let $\mathcal{T}$ be a distribution of tasks. Each task $\tau \in \mathcal{T}$ corresponds to a joint distribution $p_\tau(x, y)$. During pretraining:

1. A task $\tau \sim \mathcal{T}$ is sampled.
2. A dataset $D \sim p_\tau(x, y)$ is generated.
3. The model is trained to predict labels for new inputs conditioned on D

This defines a prior over possible tabular classification functions. Considering this, predictions can be interpreted as averaging over possible tasks:

$$p(y_*|x_*, D) = \mathbb{E}_{\tau\sim\mathcal{T}}[p_\tau(y_*|x_*, D)]. \tag{3.17}$$

The model considers many possible explanations for how the data could have been generated, and gives more weight to the explanations that fit the data well, combining them all into a final prediction.

The synthetic tasks used to train TabPFN are generated from randomly sampled structural models. These models typically include:

- Random feature interaction

- Nonlinear decision boundaries
- Different numbers of informative and noisy features
- Varying levels of class imbalance
- Different noise strengths

By training on a large number of such synthetic datasets, the model learns general patterns that commonly appear in tabular datasets. This process does not aim to replicate any specific real-world dataset. Instead, it aims to create a broad and diverse distribution of plausible tabular problems. Because the model is trained on this synthetic distribution, it develops expectations about how features relate to targets, the typical amount of noise, the strength of interactions between variables and the complexity of decision boundaries. This implicit knowledge acts as a prior when the model is applied to a new real-world dataset. The model combines this prior knowledge with the observed training examples to make predictions. Thus, the synthetic task distribution serves the same conceptual role as a prior in Bayesian inference. However, instead of being defined analytically, it is defined procedurally through data generation.

3.8.3 In-Context Inference

A defining characteristic of TabPFN is that it performs in-context inference. Instead of adapting its parameters to each new dataset through gradient-based training, the model conditions directly on the dataset with a single forward pass. This distinguishes TabPFN from traditional ML methods and connects it to Transformer-based learning.

Usually in supervised learning, a model is trained on a dataset D by optimising its parameters. After training, the dataset is no longer part of the model's input and only new feature vectors are provided during inference. TabPFN works by taking the entire training dataset as input at inference time:

$$[(x_1, y_1), ..., (x_n, y_n), (x_*, ?)].$$

The model then predicts the target y_* for the point x_*. The Transformer learns a mapping of the form:

$$q_\phi(D, x_*) \approx p(y_*|x_*, D), \tag{3.18}$$

where q_ϕ is the learned predictive mapping and ϕ are the pretrained weights. Thus, adaptation to a new dataset happens through conditioning on the input context rather than through parameter updates. This is enabled by the Transformers self-attention. Each element in the input can attend to all others. This allows x_* to be influenced by all training samples simultaneously. Practically, this means that the model can identify which training samples are most relevant for the inference, capture complex relationships between features, and adapt its predictions depending on the dataset size, noise level or class imbalance. Because self-attention scales quadratically with the number of data points, this mechanism is particularly suited for smaller datasets.

3.8.4 Comparison to Traditional Methods

TabPFN differs substantially from traditional tabular learning methods in how it is trained, how it performs inference, and how it incorporates prior knowledge.

Gradient boosting methods, such as XGBoost and CatBoost, are among the strongest baselines for tabular data. In contrast to these models, TabPFN does not train a model separately for each dataset. Instead, it has been pretrained on a large distribution of synthetic tasks and performs inference in a single forward pass without dataset-specific optimisation. Key differences include:

- **Training procedure** - Gradient boosting requires training from scratch on every dataset. TabPFN requires no additional training at inference time.
- **Hyperparameter tuning** - Boosting methods often require tuning of hyperparameters in order to perform well. TabPFN typically does not require task-specific hyperparameter optimisation.
- **Computation time** - Boosting can be computationally expensive for large hyperparameter searches. TabPFN shifts most computational cost to pretraining and is fast at inference.

TabPFN also requires less preprocessing of features than other methods and generally handles missing values well, without explicit imputation. However, boosting methods typically scale better to large datasets, whereas TabPFN is limited by the quadratic complexity of self-attention in the number of data points.

3.9 SHAP for Model Explainability

Shapley Additive Explanations (SHAP) is an explainability framework for interpreting the predictions of machine learning models. It is based on Shapley values from game theory and quantifies the contribution of each input feature to a model's output prediction.

Each input feature is treated as a "player" and the prediction as the collective "payoff". The Shapley value of feature j represents the average marginal contribution to the prediction across all possible feature subsets:

$$\phi_j = \sum_{S \subseteq \mathcal{N} \setminus \{j\}} \frac{|S|!(|\mathcal{N}| - |S| - 1)!}{|\mathcal{N}|!} [f(S \cup \{j\}) - f(S))], \tag{3.19}$$

where $\mathcal{N}$ is the full set of features and $f(S)$ is the model's expected output given only the feature subset S [27]. A key property of SHAP is that the model prediction can be expressed as a sum of feature contributions relative to a baseline value:

$$f(x) = \mathbb{E}[f(x)] + \sum_j \phi_j, \tag{3.20}$$

where $\mathbb{E}[f(x)]$ represents the expected model output over the dataset.

The SHAP framework is based on several theoretical properties derived from Shapley values. These include:

- *Efficiency*, meaning that the sum of feature contributions equals the model prediction.
- *Symmetry*, where features that contribute equally receive equal attribution.
- *Dummy*, ensuring that features with no influence receive zero contribution.
- *Additivity*, which guarantees consistency when combining multiple models [28].

SHAP supports both local and global interpretability. Locally, SHAP values decompose an individual prediction, ranking each feature's contribution to the specific sample output. Globally, the values can be aggregated across multiple instances in order to reveal the importance of each feature overall.

In practice, the exact computation of Shapley values is computationally expensive, as it requires evaluating all possible feature subsets. This results in exponential complexity with respect to the number of features. In order to address this, several approximation methods and model-specific implementations have been developed. For example, KernelSHAP provides a model-agnostic approximation using weighted linear regression[27], while TreeSHAP enables efficient exact computation for tree-based models [29]. Variants such as DeepSHAP extend the framework to neural networks.

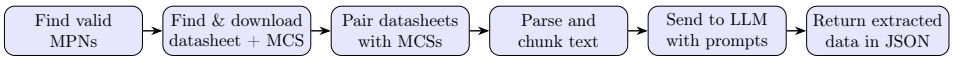
Despite its strengths, SHAP has several limitations. The method typically assumes feature independence when estimating $f(S)$, which can lead to unrealistic feature combinations and unreliable attributions in the presence of correlated features. Additionally, SHAP explanations depend on the choice of background dataset used to estimate the expected value, which can influence the resulting interpretations. The computational cost can also become significant for high-dimensional datasets.

Finally, it is important to note that SHAP reveals the statistical influence of the features rather than causal relationships and therefore does not reflect the true underlying causal mechanisms [27].

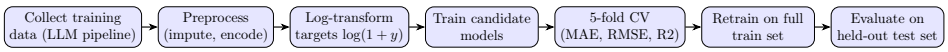
Chapter 4
Methodology

4.1 Overview

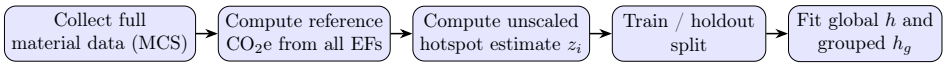
Below follows a flowchart of how different parts of the implementation work.



(a) LLM leveraged data collection pipeline



(b) Hotspot metal predictor model development pipeline



(c) Hotspot scaling factor estimation pipeline

Figure 4.1: Overview of the workflow.

4.2 Data Collection

A large, structured dataset of component parameters and respective material contents was needed in order to gather insights into how a component's physical parameters (pin count, package size, mass) related to the mass of individual hotspot metals in the components. Since no such dataset was available to us at Volvo, it had to be constructed using information publicly available from manufacturers' (e.g., Infineon and NXP) websites.

4.2.1 Finding MPNs

In order to find documentation for a specific component, the manufacturer's part number of said component was needed. In this case, the goal was a large dataset of component data, and therefore a large number of MPNs was necessary. No such comprehensive list of MPNs was available to us, and thus, we needed a way to

automatically find and collect viable MPNs that could then be used when querying for datasheets and MCSs.

The solution for this relied on automated web-based extraction and validation scripts tailored to each manufacturer's product website. Custom scripts were developed for each manufacturer to query publicly accessible websites, retrieve candidate MPN entries from search results and product listing pages. The extracted part numbers were then normalised and deduplicated to ensure consistency and to prevent repeated processing of identical records. The resulting dataset was written to structured output files incrementally, allowing the collection process to be reproducible, scalable, and resilient to interruptions during long extraction runs. The scripts mainly utilised **Playwright** to automate the web searching in Python.

4.2.2 Finding PDFs

The identification and downloading of matching PDF documents was carried out through an automated retrieval pipeline applied to each previously collected MPN. For each part number, the scripts first generated the corresponding manufacturer product-page URL and then used browser automation to access the page, handle dynamic page elements such as cookie banners, and inspect the document section for links to key files, particularly datasheets and material-content documentation. Where direct retrieval from the product page was unsuccessful, the process employed fallback search routines to locate the appropriate product entry and repeat the document search. Once document links had been identified, the files were downloaded automatically using HTTP requests and saved with structured filenames incorporating the MPN to avoid ambiguity and filename collisions. The workflow also included checks for missing or inaccessible files and, where necessary, conversion of manufacturer-supplied tabular compliance documents into a usable spreadsheet format. This automated approach ensured a consistent and efficient method for locating and saving the relevant technical documentation associated with each MPN.

4.2.3 Extracting Data from PDFs

The core of the data collection pipeline was the script utilising LLMs to extract structured data from the data sheet PDFs. A lot of inspiration for initial implementation ideas was taken from previous work on the subject, for example [3, 30]. This included parsing the PDFs deterministically using **pypdf** and feeding the LLMs the raw text along with an instruction prompt. The output was then forced into a JSON schema format using **Pydantic**.

4.2.4 Validation of LLM Data Extraction

In order to benchmark the performance and accuracy of different LLMs, a manually annotated validation set containing 115 samples was constructed. This included going through each component's datasheet and MCS, and annotating all relevant parameters. The annotated fields are listed in Table 4.1.

All 12 fields in Table 4.1 were manually annotated. However, only 10 fields were scored when reporting extraction accuracy: `component_number` was used solely to

Field	Unit or Description
Component number	Manufacturer part number identifier
Package type	Manufacturer package designation
Package class	Derived package category
Mass	mg
Function type	Component function category
Width	mm
Length	mm
Pin count	Number of pins
Total mass of copper	mg
Total mass of gold	mg
Total mass of palladium	mg
Total mass of silver	mg

Table 4.1: Fields manually annotated in the LLM validation set.

match each extraction to the correct validation sample, and `package_class` was excluded from scoring because it is derived downstream from `package_type`.

This enabled us to quantify the performance of our data extraction, which helped us develop a robust and reliable extraction script.

4.2.5 Prompts

The LLMs were fed two different prompts. A **system prompt**, defining and shaping the behaviour of the model, and a **user prompt**, containing the actual instructions to the LLM. The system prompt can be seen in Figure 4.2.

As can be seen in Figure 4.2, few-shot prompting (giving examples to the LLM) was used for package and function information to ensure higher accuracy. This is an example of in-context learning.

The **user prompt** can be seen in Figure 4.3.
The **user prompt** was sent to the LLM together with the parsed datasheet and material content sheet. These documents were trimmed down (only sending the first few pages of the datasheet) in order to minimise computation time and ensure that the LLM's context windows weren't exceeded.

The context window limits the number of tokens an LLM can handle and is the sum of tokens from the system prompt, user prompt and the model's output. By reading the first 10 datasheets, it was discovered that the relevant information could be found on the first few pages of the document, hence allowing trimming of excessive pages. We tried this approach, and it gave accurate results for the LLM-based data extraction (Figure 4.4).

4.2.6 Deterministic Post-Processing

In the material content sheets, the same material can appear multiple times (see Figure A.1). Since the total mass of each material present in a component was not explicitly written out, each respective material entry had to be summed.

You are an information extraction engine.
You may receive ONE or TWO documents: a datasheet and/or a material content sheet.
Return ONLY valid JSON that conforms exactly to the provided JSON Schema. Do not add extra keys.
Document availability rules:
- The material content sheet is the PRIMARY source for package_type.
- If the material content sheet is NOT available or does not mention package_type, you MAY infer package_type from the datasheet if it is explicitly stated (e.g. "PG-TO252-3-11", "PG-VQFN-32", "QFN-48", "TO-220").
- If neither document clearly states the package type, set package_type to null.
- Never guess or invent a package type.
Datasheet rules:
- Infer function_type as ONE of: power, logic, interface, MCU, memory, unknown. You may infer from context if not explicitly stated.
- Extract package_dimensions_raw ONLY if explicitly stated in the datasheet (single raw string, do not normalise).
- Extract package_type from the datasheet ONLY when the material sheet is missing.
Material content sheet rules:
- Extract manufacturer, component_number (Sales Product Name), package_type, mass_mg, and metals.
- Metals: include every row marked as noble metal or non-noble metal.
Missing data handling:
- If a field is not present in the available documents, use null (or "unknown" where allowed).
- Do NOT infer material-related fields from the datasheet.
- Do NOT infer datasheet-related fields from the material sheet.

Figure 4.2: System prompt used for structured information extraction from component documentation.

Extract structured data from the provided documents.
Use the material content sheet as the primary source.
If the material sheet is missing, extract what you can from the datasheet only.

Figure 4.3: User prompt for the structured data extraction.

To ensure reliable summation, materials with the same name had their masses summed deterministically. This reduced the workload of the LLM and also ensured correct calculations, since LLMs sometimes struggle with mathematical tasks.

4.2.7 Model Selection

LLMs considered for the data extraction had to be free to use, open source, small enough enough to run inference on a laptop in a reasonable time, yet large enough to perform accurately and reliably.

The three models tested were:

- Gemma 3 4B [31]
- Llama 3.1 8B [32]
- gpt-oss 120B [33]

These models were run on the validation data, with extraction accuracy and extraction time being measured.

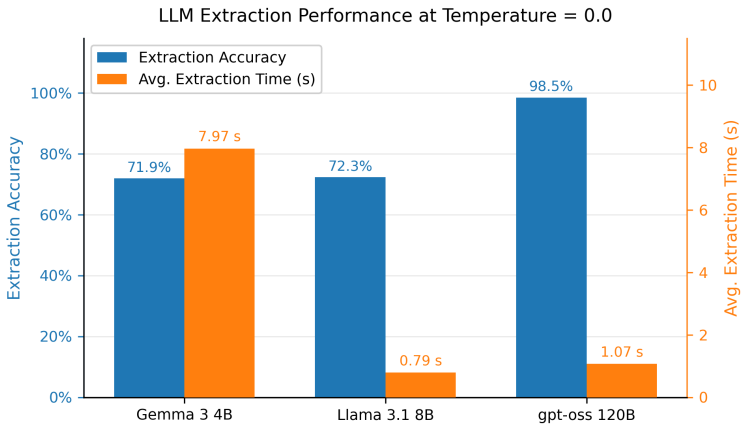


Figure 4.4: Extraction accuracy and time for the different LLMs.

Gpt-oss 120B was chosen to be used in the data extraction pipeline, due to its superior performance, as can be seen in Figure 4.4.

4.2.8 Collecting the Training Data

The data collection pipeline described above was utilised to construct a training dataset to be used for the hotspot metal predictor. 1794 samples of Infineon and NXP components were collected. In some cases, the data was incomplete, containing missing features for some samples. This was deemed acceptable since we wanted a model to be able to handle sparse or incomplete data. A sample set for two collected components can be observed in Table 4.2.

Field	Infineon 2N7002 H6327	NXP FS32K116LAT0MFMT
manufacturer	Infineon Technologies AG	NXP Semiconductors
component_number	2N7002 H6327	FS32K116LAT0MFMT
package_type	PG-SOT23-3-5	HVQFN32
package_class	SOP	QFP
mass_mg	9.32	93.091257
function_type	power	MCU
width_mm	1.3	5
length_mm	2.9	5
pin_count	3	32
metal_total_copper_mg	3.008	22.042
metal_total_gold_mg	0.005	0.009
metal_total_palladium_mg	0	0.072
metal_total_silver_mg	0.284	0.112

Table 4.2: Example component records from the collected training data.

4.3 Hotspot Metal Predictions

As previously mentioned, Zhang et al. [2] identify the so-called *hotspot metals* as being mainly responsible for the raw-material carbon emissions in semiconductor IC components. These included copper, silver, gold and palladium. The amount of each of these metals in a component can be found in the component's MCS. However, in the case that the MCS is not available, the metal weights have to be predicted using known parameters of the component (i.e., mass, pin count, package class, etc.). The metal mass prediction model presented in Zhang et al. [2] was based on linear regression, assuming a linear relationship between input features and predicted output. This made the performance very weak for unseen samples. An enhanced model for predicting hotspot metal weights was deemed necessary.

Because of the tabular dataset with both categorical and numerical features, as well as the assumed nonlinear relationships at play, tree-based Gradient Boosted models were hypothesised to perform well. Because of the fairly small dataset, with some missing entries, TabPFN was also expected to be able to perform well.

TabPFN was accessed through the TabPFN GitHub repository [34]. Open-source libraries for gradient boosted ensemble learning were used, in particular XGBoost [35] and CatBoost [36]. A lot of helper algorithms and performance metrics functions were accessed through the **sklearn** framework.

4.3.1 Data Preprocessing

After collecting the dataset using the LLM pipeline, it was prepared for model training and evaluation. The raw dataset initially contained 1794 component entries from Infineon and NXP. An initial inspection revealed duplicate entries sharing the same manufacturer part number (MPN), likely due to overlap between data sources. These duplicates were removed by retaining only the first occurrence of each unique component number, which reduced the dataset to 1443 entries.

Further inspection of the dimensional variables revealed several physically

implausible values, including width values as large as 5284 mm and a length of 1314 mm. Since such values are impossible for semiconductor packages, outlier removal was done in two sequential steps. Physical bounds were applied before statistical anomaly detection. The order of this was important because if all members of a package group contain a similar extraction error, the group-wise IQR may appear internally consistent and fail to detect the problem. Removing physically impossible values first ensures that the subsequent within-group analysis is based on a cleaner distribution.

In the first step, a hard bound of $[0.1, 100]$ mm was applied to both width and length. These limits were chosen conservatively to exclude only clearly impossible values while retaining all plausible components. This step removed 34 components.

In the second step, a Tukey IQR fence with $k = 1$ was applied independently to width and length within each package type, considering only package types with at least four observations. The group-wise approach was motivated by the fact that package type strongly constrains nominal component dimensions; components within the same package family are therefore expected to have similar body sizes, and outliers within a group are more likely to represent data errors than legitimate variation. This step removed an additional 13 components.

In total, 47 components were removed across the two cleaning steps, giving a final cleaned dataset of 1395 entries. Package type was retained during cleaning because it defined the grouping for the within-group outlier analysis, but it was excluded as an input feature during model training. Likewise, the manufacturer and MPN fields were excluded from the feature set to make the final models less dependent on identifiers that are unlikely to repeat in future use.

The dataset was split into a 90/10 train-test split. The 90% training portion was used for all model development and hyperparameter tuning, while the remaining 10% was held out for the final test evaluation.

For all ML models except TabPFN, missing numerical values were imputed with the median of each feature. Missing categorical values were imputed with the most frequent category in each feature, and categories were then converted into integers. The TabPFN framework used had automatic handling of missing values and encoding of categoricals, and thus did not need explicit preprocessing in this manner.

All target variables were non-negative metal weights measured in mg. In the main modelling setup, the regression targets were transformed as $y' = \log(1 + y)$, and predictions were mapped back to the original scale as $\hat{y} = \exp(\hat{y}') - 1$. The added constant preserves zero values while compressing the long right tail of the distribution. To evaluate whether this choice was beneficial, each model family was also trained using raw target values in mg. The evaluation protocol itself was kept unchanged across all four models: hyperparameter tuning was performed only within the 90% training split using 5-fold cross-validation, after which the selected configuration was retrained on the full 90% training split and evaluated once on the held-out 10% test split.

4.3.2 Model Development

TabPFN

The first attempted model was to combine four different TabPFN regressors, one for each hotspot metal. To ensure robust and unbiased model training and evaluation, 5-fold cross-validation was employed. The targets (metal weights) were transformed using $\log(1 + y)$. Lastly, TabPFN was tested on the holdout test set. The performance metrics of this model can be seen in Table 5.6.

TabPFN – Zero Hurdle Model

The presence of gold and palladium in the dataset was much scarcer than that of copper and silver, meaning the model could benefit from first having to classify whether a metal was present or not, and if it was, perform regression for the actual weight. A two-stage modelling approach, outlined in Algorithm 1, was implemented to handle zero-inflated target variables, combining classification and regression within a 5-fold cross-validation framework. The dataset was split using shuffled K-fold cross-validation to ensure robustness, and the process was repeated for each target variable independently. Lastly, the zero-hurdle TabPFN was tested on the holdout test set.

Algorithm 1 TabPFN Zero-Hurdle Model

$\tau^* \leftarrow$ tune threshold via F_2 on OOF probabilities from K-fold CV
for each target $y \in \{\text{Au, Pd, Ag, Cu}\}$ **do**
    $z_i \leftarrow \mathbf{1}[y_i > 0]$ for all i ▷ Binarise target
    Stage 1: Classification
    Train `TabPFNClassifier` on $(X_{\text{train}}, z_{\text{train}})$
    $\hat{p} \leftarrow P(z = 1 \mid X_{\text{test}})$ ▷ Predicted probability of nonzero
    $\tau^* \leftarrow \arg\max_\tau \ F_2(\mathbf{1}[\hat{p} \geq \tau], z)$ ▷ Threshold tuned on OOF probabilities
    Stage 2: Regression on positives
    $X^+, y^+ \leftarrow$ rows where $y_{\text{train}} > 0$
    Train `TabPFNRegressor` on $\left(X^+, \log(1 + y^+)\right)$
    $\hat{y}^+ \leftarrow \exp\left(\hat{f}(X_{\text{test}})\right) - 1$ ▷ Predict and back-transform
    Combine stages
    for each test sample i **do**
        if $\hat{p}_i \geq \tau^*$ **then**
            $\hat{y}_i \leftarrow \hat{y}_i^+$
        else
            $\hat{y}_i \leftarrow 0$
        end if
    end for
end for

Model performance was evaluated on each fold using mean absolute error (MAE), root mean squared error (RMSE), and R^2, with mean metrics aggregated across folds for each target and can be seen in Table 5.7.

XGBoost

A multi-output regression approach based on XGBoost was implemented to model all target variables simultaneously within a unified pipeline. The preprocessing steps were combined with a **MultiOutputRegressor** wrapper around an **XGBRegressor**, enabling independent gradient boosting models to be trained for each target while sharing the same feature transformation pipeline. To address skewed target distributions, a **TransformedTargetRegressor** was applied, using the $\log(1 + y)$ transformation during training and the corresponding inverse mapping during prediction to improve stability and performance on non-negative, right-skewed outputs. Model hyperparameters were optimised using **sklearn's RandomizedSearchCV** with 100 iterations over a predefined search space of XGBoost hyperparameters such as the number of estimators, learning rate, tree depth, subsampling ratios, regularisation terms, and column sampling rates. A 5-fold shuffled cross-validation strategy was used to evaluate candidate configurations based on negative mean absolute error. The best-performing model was retrained on the full training set and subsequently evaluated on the held-out test set. Performance was assessed separately for each target variable using MAE, RMSE, and R^2 metrics. The metrics can be seen in Table 5.8.

CatBoost

The same methodology as for XGBoost, including the $\log(1 + y)$ target transformation, was implemented for CatBoost. The performance metrics can be seen in Table 5.9.

4.3.3 Model Selection

For all models, the same protocol was used. First, the dataset was divided into a 90/10 train-test split. The 90% training portion was then used for model development and hyperparameter tuning via 5-fold cross-validation, meaning that in each fold 80% of that 90% portion was used for fitting and 20% for validation. After the hyperparameters had been selected, each model was retrained on the full 90% training portion and finally evaluated on the held-out 10% test split. The final held-out test metrics of the four retrained models were then compared in order to choose which model to use in the final implementation of the CO_2e predictor.

As can be seen in Tables 5.6 to 5.9, both TabPFN-based implementations performed better than the Gradient Boosted models when it came to predicting the copper weights. However, since gold has by far the largest impact on carbon emissions [2], getting an accurate gold weight prediction was deemed the most important. This led to the choice of using CatBoost as the final metal weight predictor.

4.4 SHAP

SHAP was employed in order to provide interpretability of the chosen CatBoost regression models. This was done in an attempt to gain insight into how each input feature contributes to the individual metal contents. This could potentially help guide model development, but also provide insights for the electrical cost engineers on how electronic components' physical parameters are related to carbon driving metals.

The SHAP framework [37] was applied, utilising **TreeExplainer** because of its efficiency for tree-based ensemble methods. Primarily, global interpretability was of interest. This was achieved through beeswarm summary plots, which reveal feature importance and directional effects across the dataset. Quantitative comparison was enabled by aggregating mean SHAP values per feature.

4.5 Hotspot Scaling Factor

Once the ML models had been trained and evaluated for the hotspot metal masses, the results had to be translated into CO_2e values. In Zhang et al. [2], this was done by combining predicted metal masses with metal emission factors and a hotspot parameter. That hotspot parameter was component-specific, depending on whether the component contained gold and on its package class.

For this thesis, a single global scaling factor was preferred for simplicity and better generalisation across components. The approach used the same four hotspot metals (copper, gold, palladium, and silver) and introduced one global scaling parameter h. These four metals, together with h, were used to approximate the total material-based emissions of a component. The goal was to evaluate how much information was retained when reducing the full material composition to only a few hotspot metals and one global scaling constant. In practical terms, h acts as a correction factor that maps the CO_2e explained by the four hotspot metals to the component's total raw-material CO_2e.

With a small adjustment of the prompts and JSON-schema definition, the data collection pipeline was adjusted in order to allow the collection of all available material weights in a certain component, not only the four hotspot metals as had previously been done. This was used to build a dataset of a large number of components with all of their respective material contents. Because this extension was carried out later in the project, after additional MPNs had been identified, the dataset used to estimate h became larger than the earlier dataset used for hotspot-metal prediction.

Let y_i denote the reference (ground-truth) raw-material CO_2e for sample i in g CO_2e, and let z_i denote the unscaled hotspot estimate in g CO_2e obtained from the four hotspot metals and their emission factors after the final kg-to-g conversion described below. The global no-intercept model is

$$\hat{y}_i = hz_i.$$

In this scaling-factor analysis, z_i was computed from the ground-truth hotspot masses extracted from the material-content data rather than from ML-predicted

hotspot masses, in order to isolate how well the four hotspot metals explain total raw-material CO_2e. For the grouped models, a separate scaling factor was fitted for each group, such as each package class or function type. If sample i belongs to group $g(i)$, its prediction is given by

$$\hat{y}_i = h_{g(i)} z_i,$$

where z_i is the unscaled hotspot estimate, $h_{g(i)}$ is the scaling factor for the group of sample i, and $\hat{y}_i$ is the predicted raw-material CO_2e.

All scaling factors were estimated using only the training split. For the global model, a single scaling factor h was fitted across all training samples:

$$h = \frac{\sum z_i y_i}{\sum z_i^2}.$$

For the grouped models, one scaling factor was fitted separately within each group g:

$$h_g = \frac{\sum_{i \in g} z_i y_i}{\sum_{i \in g} z_i^2}.$$

Holdout performance was evaluated over all test samples:

$$\mathrm{MAE} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} |y_i - \hat{y}_i|, \qquad \mathrm{RMSE} = \sqrt{\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (y_i - \hat{y}_i)^2}.$$

Equivalently, if the test set is divided into groups g, the overall MAE can be written as

$$\mathrm{MAE} = \frac{\sum_g \sum_{i \in g} |y_i - \hat{y}_i|}{N_{\text{test}}},$$

which shows that the errors were summed across all test samples and then divided by the total number of test samples.

In the Results chapter, the hotspot-scaling tables report both the overall holdout result and the per-group breakdowns for package class and function type. The resulting scaling factors and their holdout performance are presented in Section 5.5.

Table 4.3 lists the set of materials used in this analysis. Each material has a corresponding material emission factor. The emission factors presented by Zhang et al. [2] were used.

In addition to the global model, grouped variants based on package class and function type were evaluated in order to determine whether a small increase in model complexity could improve holdout performance.

Material	CO_2e Emission Factor (kg CO_2e/kg)
Au	52 360
Cu	4.26
Fe	133.7
Zn	238.7
Ag	987
Ni	8.07
Pd	7910
Pb	1.15
Sb	16.17
Ca	0.21
Sn	1.63
Cr	20.65
Ti	11.87
Mg	4.34
Si	20.3
SiO_2	3.5
Talc	87.5
Carbon Black	166.6
Resin	0.05022
Acrylate	1.47

Table 4.3: CO_2e emission factors for materials. Due to confidentiality the displayed values have been transformed.

4.6 Calculating the CO_2e

The CO_2e emissions were calculated as a sum of the CO_2e emissions from raw materials and the manufacturing process. Since this thesis only focuses on the raw materials part, the calculations for the process emissions were taken directly from Zhang et al. [2]. Thus, the CO_2e emissions for a certain component were calculated as:

$$\begin{aligned} E_{\mathrm{CO_2e}} &= E_{\mathrm{materials}} + E_{\mathrm{process}} \\ &\approx h \cdot E_{\mathrm{hotspot}} + E_{\mathrm{process}} \\ &= h \cdot \underbrace{\left(EF_{\mathrm{Au}} \cdot m_{\mathrm{Au}}^{(\mathrm{kg})} + EF_{\mathrm{Pd}} \cdot m_{\mathrm{Pd}}^{(\mathrm{kg})} + EF_{\mathrm{Ag}} \cdot m_{\mathrm{Ag}}^{(\mathrm{kg})} + EF_{\mathrm{Cu}} \cdot m_{\mathrm{Cu}}^{(\mathrm{kg})} \right)}_{E_{\mathrm{hotspot}}} + E_{\mathrm{process}} \end{aligned} \tag{4.1}$$

Where EF_i are the emission factors for the hotspot metals and m_i are the predicted weights of each hotspot metal in mg. Before multiplication by EF_i, each m_i is converted to kilograms according to $m_i^{(\mathrm{kg})} = 10^{-6} m_i^{(\mathrm{mg})}$.

4.7 The CO_2e Calculator Tool

Using the **Tkinter** [38] framework for Python, an app with a GUI was developed to be used by the cost engineers. The app bundled the functionalities for data collection, metal prediction and CO_2e calculations that had previously been developed. The app was developed with suggestions and feedback on functionality from the electrical cost engineering team.

Chapter 5
Results

5.1 LLMs for Data Collection

The evaluation metrics for the three different LLMs considered can be seen in Table 5.1. The extraction accuracy was calculated by running the data extraction pipeline on the datasheets and MCSs of the components in the validation dataset and counting the number of data fields in the extracted dataset that were exactly the same as in the manually annotated validation dataset, then dividing by the total number of data fields in the entire validation set. Using Python's **time** module, a timer was started right before making the LLM call and stopped after getting a successful LLM response. Thus, the PDF parsing, chunking, prompt construction and the deterministic post-processing were not included in the calculated extraction time. The extraction time was then averaged over all individual LLM calls.

Gpt-oss had the best accuracy, and only slightly longer average extraction time than Llama 3.1. Gemma 3, however, had an excessively long average extraction time. Performance proved similar across different temperatures, except for Llama 3.1, which had significantly worse accuracy at the highest temperature setting. Notably, the accuracy of Gemma 3 4B was very similar to that of Llama 3.1 8B, while having half the number of parameters.

Model Name	No. Parameters	Context Window Length	Temperature	Extraction Acc.	Avg. Extraction Time (s)
Gemma 3 4B	4 billion	128 000 tokens	0.0	0.719130	7.966
			0.5	0.695652	6.382
			1.0	0.709565	6.043
Llama 3.1 8B	8 billion	128 000 tokens	0.0	0.723478	0.792
			0.5	0.674783	0.782
			1.0	0.468696	1.037
gpt-oss 120B	116.8 billion	128 000 tokens	0.0	0.985215	1.071
			0.5	0.980000	1.083
			1.0	0.982609	1.124

Table 5.1: Results of test runs with different LLMs and temperatures.

In Table 5.2, the accuracy of extracting each particular data field for each model is shown. These were all for temperature 0.0, since that was the best setting across all models according to Table 5.1. Function type proved to be a particularly difficult field to extract correctly, with considerably lower accuracy across all models. Mass had perfect accuracy for all three models.

The peak memory usage for the three models can be seen in Table 5.3. In

Table 5.4 a memory-normalised accuracy is presented, showing extraction accuracy divided by the peak memory of the model. Gemma 3 4B has by far the highest normalised accuracy, which aligns with Table 5.1, where Gemma 3 had nearly the same accuracy as Llama 3.1, but with far fewer parameters.

Field	**Gemma 3 4B**	**Llama 3.1 8B**	**gpt-oss 120B**
function_type	0.1909	0.6538	0.9391
package_type	0.6273	0.6923	0.9739
length_mm	0.6273	0.6923	0.9739
width_mm	0.6364	0.7019	0.9739
metal_total_copper_mg	0.8273	0.5192	0.9913
metal_total_silver_mg	0.7545	0.8462	1.0000
metal_total_gold_mg	0.9455	0.9423	1.0000
metal_total_palladium_mg	0.9182	0.9808	1.0000
pin_count	0.9909	0.9712	1.0000
mass_mg	1.0000	1.0000	1.0000

Table 5.2: Accuracy for each extracted field for each model (temperature = 0.0).

Model	**Parameters**	**Quantization**	**Memory (GB)**	**Source**
Gemma 3 4B	4B	Q4_0 (QAT)	∼2.6	[39]
Llama 3.1 8B	8B	Q4_K_M	∼5–7	[40]
gpt-oss 120B	117B	MXFP4 (native)	∼61–80	[33, 41]

Table 5.3: Peak memory requirements (model weights only) for the three LLMs used. KV cache overhead is excluded.

Model	**Accuracy (temp = 0.0)**	**Memory (GB)**	**Accuracy/GB (%/GB)**
Gemma 3 4B	71.9%	∼2.6	27.7
Llama 3.1 8B	72.3%	∼5–7	10.3–14.5
gpt-oss 120B	98.5%	∼61–80	1.2–1.6

Table 5.4: Memory-normalised accuracy of the three candidate LLMs at temperature 0.0.

5.2 Extracted Data

A Pearson correlation matrix shown in Figure 5.1 was used to evaluate linear relationships between the dataset columns. This gives an indication of how strongly

each target depends on each feature and provides a baseline for comparing non-linear relationships captured by the ML models.

Figure 5.1: Correlation matrix of the cleaned dataset.

Mass and copper showed a strong positive correlation (0.93). Copper content, therefore, closely tracks total component mass, which is expected. Pin count also showed a strong positive correlation with silver and a notable positive correlation with gold.

Overall, mass and pin count appear to be the primary drivers of metal content among the analysed features. These relationships are likely interconnected, meaning that components with more metal pins tend to weigh more, and heavier components tend to contain more copper, silver, and gold. At the same time, the negative correlation of -0.16 between pin count and copper indicates that higher pin counts may shift the composition toward other conductive metals.

	count	mean	std	min	25%	50%	75%	max
mass_mg	1395	635.62	853.28	0.21	100.09	321.69	856.61	6908.09
width_mm	1340	8.19	5.23	0.60	4.00	7.00	10.16	40.00
length_mm	1340	9.23	5.34	0.32	5.34	8.55	12.80	40.00
pin_count	1395	54.18	95.98	2.00	3.00	8.00	64.00	541.00
metal_total_copper_mg	1395	282.67	498.68	0.00	31.35	87.28	234.19	4748.84
metal_total_gold_mg	1395	0.38	2.46	0.00	0.00	0.001	0.04	28.55
metal_total_palladium_mg	1395	0.01	0.04	0.00	0.00	0.00	0.007	1.24
metal_total_silver_mg	1395	1.60	2.68	0.00	0.11	0.48	1.86	37.98

Table 5.5: Summary statistics of the cleaned dataset.

Inspection of the distribution statistics showed that gold and palladium masses are strongly right-skewed. At least 25% of all gold values and 50% of palladium values were zero. Such a skew can lead to unstable training because a small number of larger values may dominate learning. Log(1+y) scaling was therefore used in the main experiments to compress larger values and produce a more even spread, as shown in Figure 5.2 for gold and palladium. The practical effect of this choice is examined in Tables 5.11a to 5.11d.

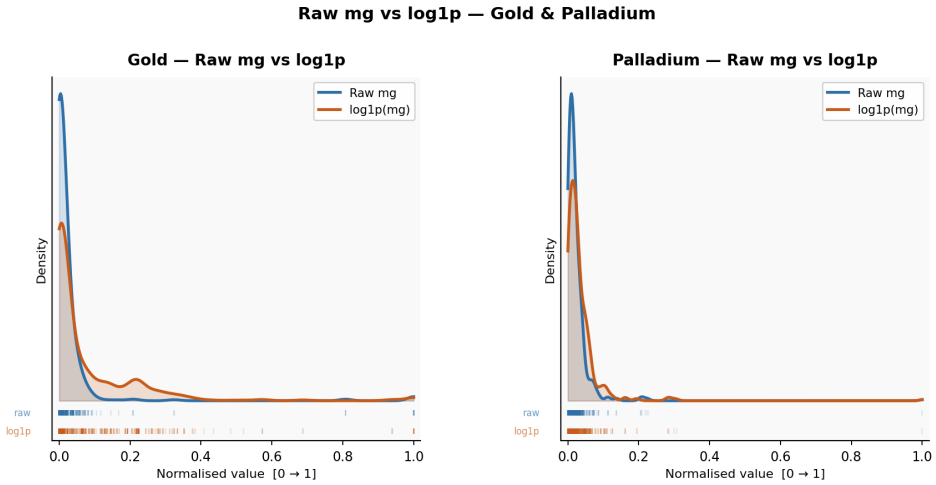


Figure 5.2: Effect of the $\log(1 + y)$ transform on the gold and palladium distributions.

5.3 Hotspot Metal Predictor Models

CatBoost provided the best results for all metals except copper. Both TabPFN implementations gave better results for the copper estimations than the Gradient Boosted models. However, they both performed worse than XGBoost and CatBoost for all other metals. Palladium was particularly difficult for the TabPFN implementations to predict, as shown by the significantly lower R^2 than for the other metals. For CatBoost, copper proved most difficult to predict, and XGBoost had the most problems with silver. The metrics in this section refer to the main $\log(1 + y)$-based training setup unless otherwise stated.

The best hyperparameters for the CatBoost model, found through the **RandomizedSearchCV** can be seen in Table 5.10.

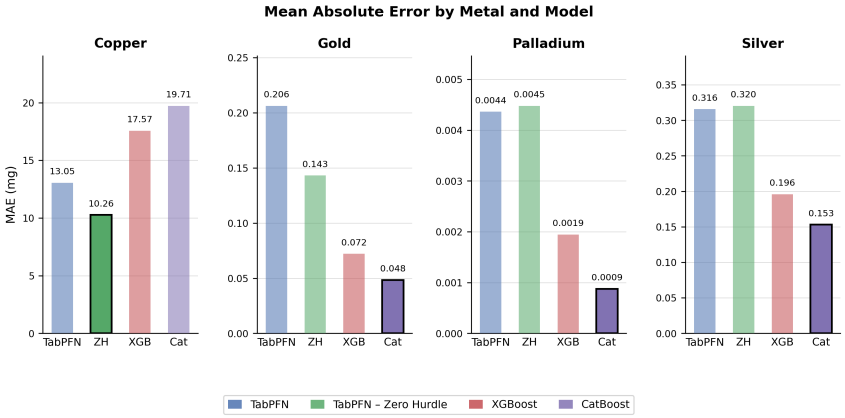


Figure 5.3: MAE for the different model implementations and metals.

Target	MAE	RMSE	R^2
Copper (mg)	13.049115	92.863486	0.954583
Gold (mg)	0.206166	1.331240	0.634851
Palladium (mg)	0.004362	0.022799	0.597897
Silver (mg)	0.315660	0.933160	0.870113

Table 5.6: Model performance metrics for the TabPFN model.

Target	MAE	RMSE	R^2
Copper (mg)	10.261504	62.617027	0.976807
Gold (mg)	0.143052	0.870355	0.844204
Palladium (mg)	0.004476	0.025484	0.451190
Silver (mg)	0.320231	0.938056	0.868871

Table 5.7: Model performance metrics for the TabPFN zero-hurdle model.

Target	MAE	RMSE	R^2
Copper (mg)	17.570329	124.837057	0.941495
Gold (mg)	0.072104	0.273309	0.985272
Palladium (mg)	0.001944	0.005541	0.943397
Silver (mg)	0.195956	0.574122	0.936562

Table 5.8: Model performance metrics for XGBoost.

Target	MAE	RMSE	R^2
Copper (mg)	19.712625	131.521573	0.935061
Gold (mg)	0.048395	0.174929	0.993967
Palladium (mg)	0.000876	0.003953	0.971191
Silver (mg)	0.153016	0.461298	0.959045

Table 5.9: Model performance metrics for CatBoost.

Parameter	Value
iterations	1000
learning_rate	0.1
depth	10
l2_leaf_reg	1
subsample	0.8
rsm	0.8
min_data_in_leaf	1
random_strength	1.0
bagging_temperature	0.0
loss_function	RMSE

Table 5.10: Best hyperparameters for CatBoost.

5.3.1 Effect of Log(1+y) Target Scaling

Since the choice of target scaling was motivated by the right-skewed distributions in Figure 5.2, each model family was also trained and tested using raw metal weights in mg. This was done to examine whether the transformation produced a measurable improvement of prediction performance.

The comparison shows that the effect of the transformation depends on both the targets and the model. For the plain TabPFN model, $\log(1 + y)$ improved copper and silver, while raw targets only improved gold slightly, and palladium was essentially unchanged. For the zero-hurdle TabPFN model, the transform was better for copper, gold, and silver, while raw targets only helped palladium slightly. The large 53 % copper MAE increase and 20 % gold MAE in the raw two-stage setting suggests that the zero hurdle model gains much more in precision when it comes to using the $\log(1 + y)$ transform over the plain TabPFN model.

For XGBoost, the clearest gain from the transform is for gold, whose MAE is 41 % worse on the raw scale, indicating that the log transform makes the highly skewed gold distribution easier to learn. CatBoost benefits the most overall, the gold MAE falls by 61 % and palladium by 4 % under the transform, while raw remains slightly better only for copper and silver. Together, these results suggest that $\log(1 + y)$ is most useful for the rare, right-skewed noble-metal targets, especially gold, because it reduces the influence of a few large observations and lets the model

(a) TabPFN

Metal	Log	Raw Δ
Copper	**13.05**	+3.04 (+23%)
Gold	0.206	**-0.007 (-4%)**
Palladium	**0.0044**	+0.000 (+1%)
Silver	**0.3157**	+0.009 (+3%)

(b) Zero-Hurdle TabPFN

Metal	Log	Raw Δ
Copper	**10.26**	+5.45 (+53%)
Gold	**0.143**	+0.028 (+20%)
Palladium	0.0045	**0.000 (-3%)**
Silver	**0.3202**	+0.009 (+3%)

(c) XGBoost

Metal	Log	Raw Δ
Copper	**17.57**	+0.52 (+3%)
Gold	**0.0721**	+0.030 (+41%)
Palladium	0.0019	**0.000 (-1%)**
Silver	0.1960	**-0.002 (-1%)**

(d) CatBoost

Metal	Log	Raw Δ
Copper	19.71	**-2.42 (-12%)**
Gold	**0.0484**	+0.030 (+61%)
Palladium	**0.00088**	+0.000 (+4%)
Silver	0.1530	**-0.006 (-4%)**

Table 5.11: Effect of $\log(1+y)$ target scaling on MAE across all model families. Raw Δ reports the change relative to the log-scaled model.

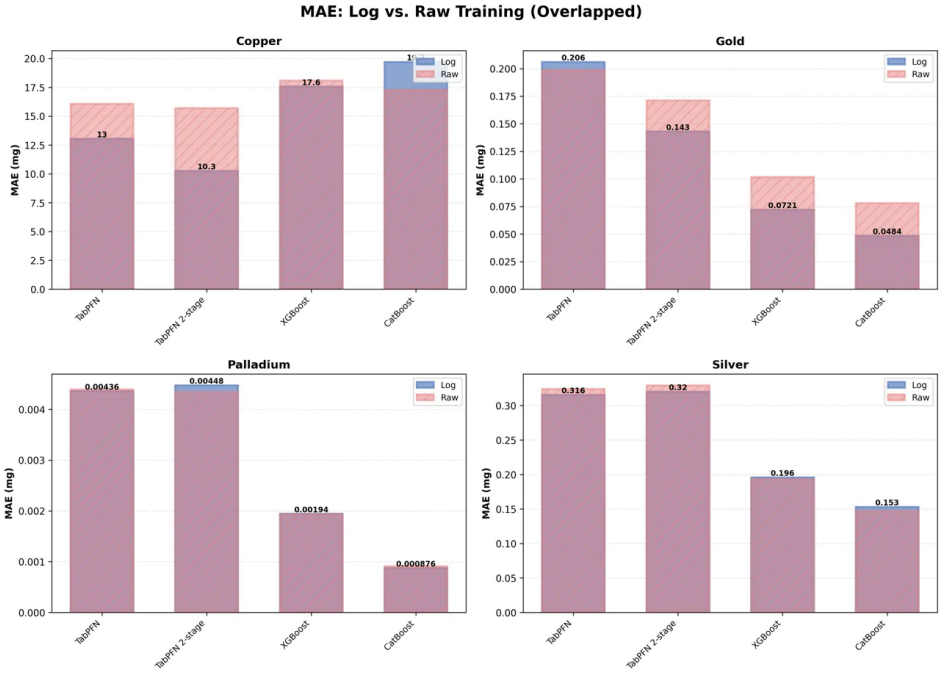


Figure 5.4: MAE per metal and model implementation with both raw and $\log(1+y)$ targets.

fit the many zero and small positive values more evenly. For smoother targets such as copper and, to some extent, silver, the raw scale can still retain useful magnitude information. The transformation should therefore be seen as a way of stabilising the hardest and most carbon-driving targets rather than as a universal improvement for all four metals [2].

5.4 Results from SHAP Analysis

As can be seen in Figures 5.5 and 5.6, the components' total mass was the most contributing factor for copper content. The number of connector pins was the most important for both gold and silver. The package class was most important for the palladium weight.

The plots in Figure 5.5 show SHAP summaries for the four different hotspot metals. Each dot represents a sample from the dataset. The rows represent the input features, ordered by importance (highest to lowest). The x-axis shows SHAP-value, i.e., how much a particular feature drove the model's predicted value up or down. The colour of the dot represents the value of the feature. For example, in the plot for copper, it is clear to see that component mass is the most important feature, with a high component mass contributing to a higher predicted copper mass and thus getting a high SHAP value. The horizontal spread of the dots shows a feature's range of influence, meaning that a tight cluster near SHAP value 0 indicates that a feature has barely any impact on the prediction.

Figure 5.6 shows a heatmap of the importance of each feature, per metal. Each column represents a metal, and each row a feature. The numbers tell the relative importance of the particular feature for the respective metals (1 being most important), with the colour of the box showing the mean SHAP value for the respective feature and metal (normalised per column).

Table 5.12 and Figure 5.7 show the mean absolute SHAP values for each metal and feature. Since the models were trained on $\log(1 + y)$-transformed targets, the SHAP values are expressed in the same log-scale, meaning that, for example, mass shifts the predicted $\log(1 + m_{\mathrm{Cu}})$ by 0.88636 on average in absolute terms across the dataset.

Feature	Copper	Gold	Palladium	Silver
mass_mg	0.88636	0.06603	0.00469	0.15523
package_class	0.18570	0.06714	0.00543	0.07538
width_mm	0.15924	0.02740	0.00309	0.08184
pin_count	0.14595	0.07431	0.00460	0.20182
length_mm	0.13882	0.05270	0.00185	0.08368
function_type	0.07026	0.01948	0.00204	0.07624

Table 5.12: Mean SHAP value per feature and metal (log-scale).

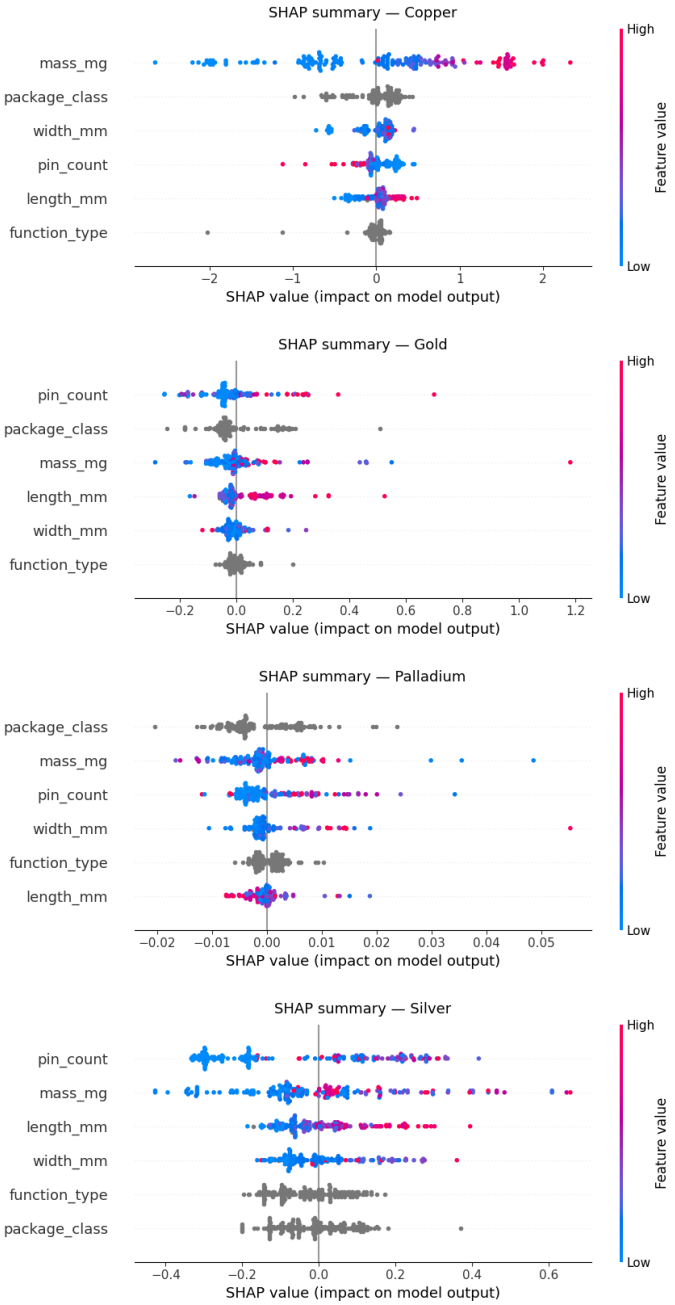


Figure 5.5: SHAP summaries for the four hotspot metals.

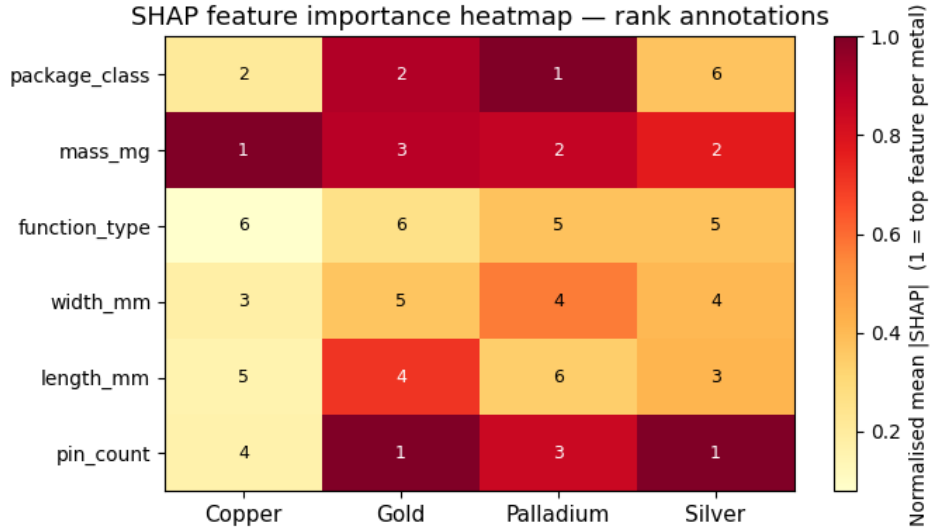


Figure 5.6: Features ranked by importance per metal.

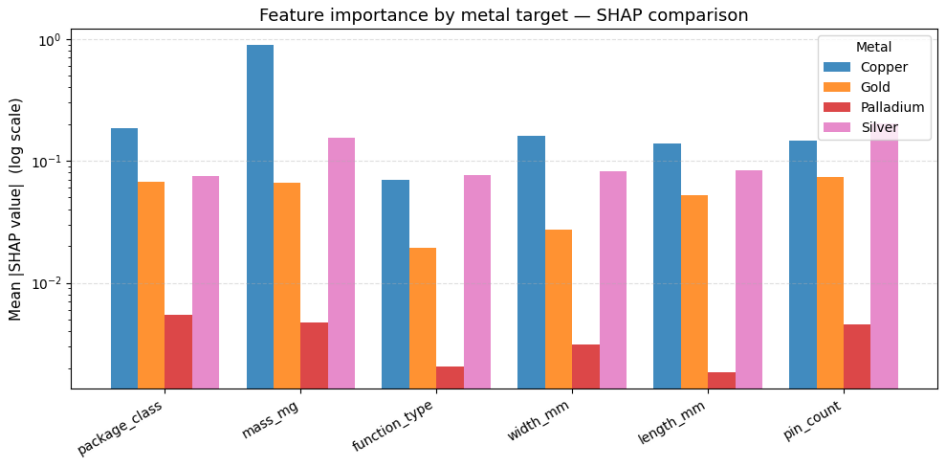


Figure 5.7: Mean SHAP value per feature for each metal.

5.5 Hotspot Scaling Factor

This section compares a single global scaling factor with grouped alternatives in order to assess how much accuracy is gained by introducing category-specific scaling. Least-squares fitting gave a global scaling parameter of $h = 1.00602$. In both Table 5.13 and Table 5.14, each row reports the number of training and test samples, the scaling factor fitted on the training split, the holdout MAE with the global factor, the holdout MAE with the group-specific factor, and the change between them.

To avoid ambiguity, the row 'ALL (test-weighted)' is neither an average of the subgroup MAEs nor the maximum subgroup error. Instead, it is the overall holdout MAE obtained by first computing one absolute error for each test component and then averaging across all test components:

$$\mathrm{MAE}_{\mathrm{ALL}} = \frac{1}{N_{\mathrm{test}}} \sum_{i=1}^{N_{\mathrm{test}}} |y_i - \hat{y}_i|.$$

This means that larger subgroups naturally have more influence on the overall result than very small ones. The subgroup rows are shown to indicate where performance improves or worsens, but model comparison should be based on the 'ALL (test-weighted)' row.

On the holdout set, package-class-specific scaling reduced the overall MAE from 2.85672 to 2.71267 g CO_2e, while function-type-specific scaling reduced it to 2.79499 g CO_2e. The package-class-specific model, therefore, gave the best overall test-set result. This value is an average over all 534 test components, not an average over package classes. The improvement is mainly driven by QFP and SOP, which together account for 447 of the 534 test components, and both improve under grouped scaling. Although DIP shows the largest relative reduction, it contains only two test samples and therefore has almost no effect on the overall MAE.

The sample counts in Tables 5.13 and 5.14 are larger than the 1395-component dataset summarised in Table 5.5 because the scaling-factor analysis was performed later on an expanded dataset with additional MPNs and full material-composition extraction, giving 2670 components in total.

For function type, `power`, `interface`, and `unknown` show lower holdout MAE with grouped scaling, `MCU` is nearly unchanged, and `logic` shows a higher holdout MAE. The overall gain is therefore modest, since the dominant MCU group accounts for 335 of the 534 test components and changes very little, while the larger relative improvements occur in much smaller groups.

Package class	n_{train}	n_{test}	h_{train}	MAE (global h)	MAE (package-class h)	ΔMAE	Relative change (%)
ALL (test-weighted)	2136	534	1.006025	2.856721	2.712673	-0.144048	-5.042419
QFP	967	250	1.030091	4.054010	3.862311	-0.191699	-4.728622
SOP	844	197	1.057522	1.472005	1.367184	-0.104821	-7.120984
BGA	320	85	1.004223	2.540335	2.509494	-0.030841	-1.214059
DIP	5	2	1.129499	3.036364	0.173584	-2.862779	-94.283148

Table 5.13: Comparison of hotspot scaling by package class (errors in g CO_2e).

Function type	n_{train}	n_{test}	h_{train}	MAE (global h)	MAE (function-type h)	ΔMAE	Relative change (%)
ALL (test-weighted)	2136	534	1.006025	2.856721	2.794991	-0.061729	-2.160847
MCU	1294	335	1.006696	3.576751	3.576297	-0.000454	-0.012695
power	672	144	1.052289	1.814310	1.666294	-0.148017	-8.158291
logic	42	23	1.001533	1.613151	1.807513	0.194362	12.048624
interface	80	22	1.034315	0.200662	0.118994	-0.081668	-40.699214
unknown	48	10	1.378407	2.449946	1.032885	-1.417060	-57.840484

Table 5.14: Comparison of hotspot scaling by function type (errors in g CO_2e).

5.6 The CO_2e Calculator Tool

A simple app was developed in order to provide an easy-to-use interface for the metal predictor pipeline. One of the goals of the thesis was to implement our findings into the electrical cost engineering workflow, reducing manual labour and providing a tool to act as a second opinion in decision-making processes. The app features a GUI where a component's MPN can be entered. The app has the collected dataset as a database, and if the component is present in the dataset, all physical parameters are returned together with the component's hotspot metal contents. The CO_2e due to raw materials and due to the manufacturing process are calculated separately, and the total CO_2e is presented to the user at electronic component level. If a component is not present in the dataset, the previously developed data collection scripts are called in order to automatically find the relevant information on the manufacturer's webpage. If metal contents can not be found, the developed CatBoost model predicts the hotspot metal contents based on the component's physical parameters (e.g., pin count, package class, function type). Components can be added to the database for future reference. All fields can also be altered manually, meaning the app can also be used as a direct CO_2e calculator, where component parameters can be entered manually, and the CO_2e emissions are calculated.

Thus, the app can both automatically find and provide component parameters along with CO_2e emissions, and be used as a calculation tool, where manually entered component parameters render approximated metal contents and CO_2e emissions for semiconductor IC components. A screenshot of the GUI can be seen in Figure B.1 in the appendix.

Chapter 6
Discussion

6.1 LLM Performance

Accuracy was the highest when the temperature was zero for all models, which is to be expected since the task and data were very deterministic. Across all models, function type was the parameter that proved most difficult to extract reliably. This can be attributed to the non-deterministic nature of function type classification. Since the datasheets do not explicitly state the function of a component as one of our predefined classes (MCU, power, logic, interface or memory), the LLMs have to infer it from the semantic context.

On the contrary, mass had perfect accuracy across all models. This can be attributed to it being very deterministic. In most cases, the mass is written out in a very straightforward manner (see the row named "Weight*" in the MCS in Figure A.1). With only one simple value in one location, the LLM has no problem with correctly extracting the mass.

The opposite effect could be seen with copper mass, which had reduced accuracy for all models as well. Copper is the most common metal of the four hotspot metals, meaning that almost every MCS will have instances of copper, and most often in more than one row. This perhaps leads to the model getting confused and neglecting previous instances of copper when faced with new ones. The results from the LLM evaluation reveal a clear pattern. Extraction difficulty is related to how explicit the target field is in the source document. The temperature results align with this interpretation. At temperature zero, the model behaves deterministically, which is well-suited to a task where the correct answer is either present in the document or it is not. The slight accuracy reductions at higher temperatures, particularly the sharp drop for Llama 3.1 at temperature 1.0, suggest that introducing randomness into generation is actively harmful when the task calls for pure extraction rather than creative inference.

An interesting finding is that Gemma 3 4B matched the extraction accuracy of Llama 3.1 8B despite having half the number of parameters. This goes against the intuition that larger models are directly better. A likely explanation is that structured extraction from well-formatted technical documents relies more heavily on instruction-following capability and training data quality than on raw model scale. Gemma 3 may have been trained with a stronger emphasis on instruction-following or structured output tasks, making it competitive here despite its smaller

size. This has practical implications for deployment in a cost-constrained or latency-sensitive setting, a smaller but well-trained model may be preferable.

The choice of gpt-oss 120B as the final extraction model was driven primarily by its substantially higher accuracy, 98.5% overall versus roughly 72% for the smaller alternatives. However, this came with a meaningful tradeoff. Initially, the idea was to get the LLM to run locally on a laptop. With 120 billion parameters, gpt-oss sits at the upper boundary of what is feasible under this constraint, and the practical viability of running such a model locally should be acknowledged as a limitation. The choice was made to circumvent this constraint by accessing the model via an API, since getting a reliable and accurate dataset was deemed the highest priority. As can be seen in Table 5.3 both Gemma 3 and Llama 3.1 can realistically be run on a common laptop, whereas gpt-oss 120B is far too large. In our experiments, Gemma 3 proved to be the most accurate model for its size, with by far the highest memory-normalised accuracy. The excessively long average extraction time for Gemma 3, as seen in Table 5.1 would conversely make Llama 3.1 the choice of model in the pipeline, if API access was to be strictly avoided.

Finally, a note on the generalisability of the pipeline. The validation set was constructed exclusively from Infineon and NXP components, both of which produce relatively well-structured and consistently formatted documentation. The example MCS in Appendix A illustrates a clean format with clearly labelled fields. The prompts fed to the LLMs were constructed with the formats of these particular documents in mind. In practice, manufacturer documentation varies considerably in format, language, and completeness. Some manufacturers use scanned PDFs with limited text quality, others omit fields entirely, and naming conventions for data fields and substances differ across the industry. The pipeline's performance on documentation from other manufacturers is therefore uncertain, and the accuracy figures reported here should be interpreted in that context. Extending the validation set to cover a broader range of manufacturers and document formats, as well as developing more general prompts, would be a valuable step before wider deployment.

6.2 Hotspot Metal Prediction Models

The results from the metal prediction models show consistent results across the different implementation approaches: gold, palladium and silver are predicted more accurately by the gradient boosted models than by any of the TabPFN implementations. Copper, however, proved more challenging for both XGBoost and CatBoost, but was handled well by TabPFN.

The results for copper could be explained in a fairly straightforward manner. As can be seen in the correlation matrix in Figure 5.1, copper mass has a strong correlation only to the total mass of the component ($r = 0.93$). This strong linear relationship means that even a small, simple model can capture most of the variance in copper by learning a nearly proportional relationship to mass. TabPFN worked well for this kind of low-complexity task, as shown by its high R^2 for copper. Gradient boosted models are optimised for non-linear relationships and may overfit to the noise in the copper mass.

Gold and palladium show a different behaviour. Both are present in very small

quantities and are highly zero-inflated; at least 25% of gold values and 50% of palladium values in the dataset are zero. This creates a different prediction problem. The model must first implicitly learn to distinguish components that contain the metal at all from those that do not, and then estimate the quantity for those that do. CatBoost achieved an R^2 of 0.994 for gold and 0.971 for palladium, which is notable given the difficulty of the task. The zero-hurdle TabPFN model was explicitly designed to handle the high zero-rate of gold and palladium by separating the classification and regression stages, yet it still performed worse than CatBoost for both metals. A possible explanation is that CatBoost's ordered boosting and native categorical handling allow it to utilise interactions between package class, pin count, and the presence of noble metals more effectively than TabPFN's in-context inference on a dataset of this size. TabPFN's strength, approximating Bayesian inference without task-specific training, may be less of an advantage when the dataset is large enough and structured enough for gradient boosting to learn the relevant interactions.

The contrastive experiment with raw targets adds a more nuanced picture. As shown in Tables 5.11a to 5.11d, $\log(1 + y)$ was not uniformly better for every model-target pair, and the effect depended both on the target distribution and on the model family. For the plain TabPFN model, the transform reduced the copper MAE from about 16.09 to 13.05 mg and the silver MAE from about 0.325 to 0.316 mg, while raw targets only improved gold slightly, from about 0.206 to 0.199 mg. Palladium changed only marginally in absolute terms, so that difference should not be over-interpreted. A plausible explanation is that the log transform stabilises broad continuous targets such as copper and silver by reducing the influence of larger observations, whereas for gold, the plain TabPFN model appears to benefit from seeing the rare larger raw values more directly rather than in compressed form.

For the two-stage TabPFN model, the transform was better for copper, gold, and silver, while raw targets only helped palladium slightly. This is consistent with the zero-hurdle design, the two-stage formulation is mainly intended for zero-inflated targets, so it is naturally better matched to gold and palladium than to copper. The 53 % MAE deterioration for copper on the raw scale corresponds to an absolute increase of 5.45 mg, from 10.26 to 15.71 mg. This is substantial, but the percentage is also amplified by the fact that the log-based copper baseline is already low. In practice, the result suggests that combining a hurdle architecture with raw copper targets gives too much influence to the largest copper values, even though copper is a dense, mostly non-zero target that is largely explained by total component mass.

For XGBoost, the benefit of the transform is clearest for gold. Raw-target training increases the gold MAE by 41 %, from about 0.072 to 0.102 mg, while the copper increase is only 3 %, from about 17.57 to 18.09 mg. This is consistent with the behaviour of boosting on skewed targets. On the raw scale, a small number of large gold values can dominate the residual updates and the tree splits and applying $\log(1 + y)$ compresses those extremes and lets the model allocate capacity more evenly across the many zero and small positive observations. Copper, by contrast, is much closer to a smooth mass-driven regression problem, so the transform still helps, but much less dramatically.

CatBoost benefits the most from the transform because it is the strongest learner on the difficult noble-metal targets once the target distribution has been regularised. For gold, the MAE drops by 61 %, from 0.0781 to 0.0484 mg, and for palladium by 4 %, from about 0.00091 to 0.00088 mg. The much larger gain for gold than for palladium is reasonable because gold combines strong right-skew with enough non-zero variation for the transform to matter, whereas palladium is so close to zero for so many samples that the absolute room for improvement is very small. Raw targets still perform slightly better for copper and silver in CatBoost, which suggests that for less skewed, more continuously varying metals, the compression can remove some useful scale information. Overall, CatBoost appears to benefit most because its ordered boosting and categorical handling can exploit the cleaner post-transform target structure particularly well, especially for gold, which is both skewed and highly important for the final CO_2e estimate [2].

Since the results show that CatBoost performed best for all metals except copper, where the TabPFN models were better, a dual model solution could be considered. CatBoost and TabPFN could be used together, where TabPFN would predict only the copper weight, and CatBoost would predict the remaining metal weights. However, as described in Section 3.8, TabPFN performs in-context inference, meaning the entire training dataset must be passed as input at every prediction. This possibly makes it less suited for online prediction in a deployed application, since the time to perform inference scales with dataset size rather than remaining constant. By contrast, once trained, CatBoost produces a fixed and compact model where new predictions require only a single pass through the learned trees, giving low and consistent latency regardless of training set size. This practical consideration, combined with the added complexity of maintaining two separate model frameworks in parallel, made the single CatBoost solution the more appropriate choice for integration into the CO_2e Calculator Tool.

6.3 Hotspot Scaling Factor

Compared with the model made by Zhang et al. [2], which used 10 hotspot scaling factors split by function type and gold content, the present results suggest that many of the same results can be retained with a substantially simpler global scaling factor. The global factor being very close to one suggests that the raw-material emissions in this dataset are already explained to a large extent by the four hotspot metals.

Since the scaling factor analysis was done using true hotspot masses and not ML-predicted hotspot masses, the linear regression only evaluates how well the hotspot approximation works on its own. If ML-predicted hotspot masses had been used instead, prediction errors from the ML model would also have affected the regression and made the estimated scaling factor h less reliable.

Grouped scaling still captures some remaining structure, but the gains are modest. Of the tested alternatives, `package_class` appears to be the more useful grouping variable, since it improves holdout performance more consistently than `function_type`. This partly contrasts with Zhang et al. [2], where `function_type`, together with whether the component contained gold, was used to determine hotspot

scaling.

At the same time, the fitted package-class factors should be interpreted with care. In particular, the large improvement for `DIP` is based on very few observations, making that estimate highly sensitive to sampling noise. Taken together, these results make the global scaling factor the most robust choice, while package-class-specific scaling appears to be the most reasonable change when a small increase in model accuracy is necessary. In practice, the reduction from 2.85672 to 2.71267 g CO_2e is modest, and whether such a gain justifies the added complexity of package-class-specific scaling is ultimately a trade-off that Volvo Cars must judge based on its desired balance between simplicity and accuracy.

6.4 SHAP

The SHAP analysis provides insights into how the CatBoost model uses each input feature, and the results are largely consistent with physical expectations for semiconductor components.

For copper, total component mass dominates. This aligns with the strong linear correlation between mass and copper content (r = 0.93) observed in the correlation matrix in Figure 5.1 and could be interpreted as copper being a bulk structural metal whose content scales nearly proportionally with component size. The relatively small contributions of the other features would suggest that copper is almost entirely dependent on the total mass, meaning even a simple model would be able to successfully capture the relationship.

For gold and silver, pin count showed to be the most important feature, as can be seen in Figure 5.6. This is physically interpretable: gold is often used in wire bonding between the die and the leadframe, while silver is commonly found in lead plating, both of which are connected to the number of connector pins. The SHAP analysis, therefore provided credible explanations beyond statistical association.

Package type also proved to have a strong importance for the gold prediction. This aligns with the findings presented by Kuo et al. [10] of different packaging types using more or less gold.

Function type consistently ranked among the least important across all four metals, as can be seen in Table 5.12. This may be surprising given that Zhang et al. [2] used function type as a grouping variable in their regression models. A likely explanation is that it is strongly linked to other features of a component. MCUs, for example, often have higher pin counts and specific package classes, meaning the model can implicitly capture the effect of function type through these correlated features. This interpretation is further supported by the hotspot scaling results, where function-type-specific scaling produced only small improvements over the global factor.

The beeswarm plots in Figure 5.5 also reveal differences in the distribution of SHAP values across metals. For copper, the horizontal spread is substantially wider than for the other metals, reflecting both the higher absolute magnitudes and the large variance in copper content across the dataset. For gold and palladium, the majority of samples cluster tightly near a SHAP value of zero across all features, which is a direct consequence of many samples containing no gold or palladium.

For these components, the model assigns near-zero feature contributions regardless of the input values.

It should be noted that SHAP values are computed under the assumption of feature independence when marginalising over absent features. Since several input features in this dataset (mass, pin count, width, and length) are physically correlated, the attribution of importance between these features should be interpreted with some care. In the presence of multicollinearity, SHAP may distribute importance somewhat arbitrarily between correlated predictors, which is a known limitation [27]. The directional effects and rankings of features reported in the results should therefore not be interpreted as definitive.

6.5 Estimated CO_2e Emission Values

A fundamental challenge in validating any carbon footprint estimation model for semiconductor components is the almost complete absence of ground-truth CO_2e values from manufacturers. Unlike material content information, which is often disclosed through material content sheets, actual CO_2e emission figures are rarely published at the component level. Manufacturers who do report emissions typically do so at a product line or facility level, combined across large volumes of components and production processes, making direct comparison with component estimates very difficult. This lack of insight is not unique to any single manufacturer but reflects a broader industry-wide lack of standardisation in component-level environmental reporting, a challenge also acknowledged in the LCA literature [8, 9]. As a result, the model developed in this thesis should not be interpreted as a tool for producing definitive or certifiable CO_2e figures, but rather as a decision-support instrument. Its primary value lies in reducing the manual effort currently required for electrical cost engineers to arrive at a rough carbon estimate, replacing a process that involves manual lookup of material data, individual calculations and significant domain expertise with an automated pipeline that produces a consistent and reproducible estimate within seconds. Used in this way, the tool acts as a second opinion, helping electrical cost engineers to quickly detect high-emission electronic components, compare alternatives and prioritise where more detailed analysis may be needed, rather than serving as a definitive truth value.

Chapter 7
Conclusions

In this thesis, it was investigated whether AI and ML techniques could be used to improve and automate carbon footprint estimation for semiconductor components, with the goal of supporting electrical cost engineers at Volvo Cars in their work. The project resulted in two main technical contributions: an LLM-based data extraction pipeline for extracting structured component data from manufacturer documentation and a set of ML models for predicting metal content, ultimately integrated into a desktop GUI application.

7.1 Data Collection Pipeline

The LLM-based extraction pipeline demonstrated that structured data can be reliably extracted from partly unstructured manufacturer documentation. Among the three models evaluated, gpt-oss 120B achieved the highest overall extraction accuracy of 98.5%, substantially outperforming the smaller alternatives. The results also showed a connection between extraction difficulty and how explicitly a field is stated in the source document, with deterministic fields such as component mass achieving perfect accuracy across all models, while semantically and contextually inferred fields, such as function type, proved much harder. An important finding was that Gemma 3 4B matched the extraction accuracy of Llama 3.1 8B despite having half the number of parameters, suggesting that model size is not the only important factor for this kind of structured extraction task. This makes Gemma 3 4B a great choice under memory restrictions. The pipeline's main limitation is its dependence on well-formatted documentation, and its accuracy on documentation from manufacturers beyond Infineon and NXP remains to be evaluated.

7.2 Hotspot Metal Prediction

The ML models developed for hotspot metal prediction showed consistent improvements over the linear regression baseline from Zhang et al. [2]. CatBoost was deemed the best model, with an R^2 of 0.994 for gold and 0.971 for palladium, the two metals of our identified hotspot metals, which contribute most to raw-material CO_2e emissions. The $\log(1 + y)$ target transformation proved to be a deliberate and worthwhile trade-off, improving prediction accuracy for the high-impact noble metals at the cost of small reductions in performance for copper and silver. TabPFN

performed competitively for copper, where the strong linear relationship with total component mass made the task well-suited to its Bayesian in-context inference. However, its inference mechanism, which requires the full training dataset to be passed as input at every prediction, makes it worse suited for deployment in a real-time application, compared to gradient boosted methods, and CatBoost was therefore selected as the sole model for the final tool. The results confirm that non-linear tree-based models are better equipped to capture the complex relationships between component features and noble metal content than the linear approach used previously.

7.3 SHAP Analysis

The SHAP analysis provided physically interpretable explanations of the CatBoost model's predictions, giving credibility to the model beyond its numerical performance alone. Total component mass emerged as the most important feature for copper content, consistent with the strong linear correlation observed in the data. Pin count was the most important feature for both gold and silver, which aligns with the known use of gold in wire bonding and silver in lead frames, both of which scale with the number of connector pins. Package class showed strong importance for palladium and gold, consistent with findings from the LCA literature that packaging choice influences noble metal usage [10]. Function type consistently ranked as the least important feature across all four metals, suggesting that its effect is largely captured through correlated features such as pin count and package class. These results support the validity of the chosen feature set and offer cost engineers a physically grounded understanding of what drives metal content in semiconductor components.

7.4 Hotspot Scaling Factor

The analysis of the hotspot scaling factor showed that a single global scaling factor of $h \approx 1.00602$ is sufficient to translate hotspot metal emissions into total raw-material CO_2e estimates with reasonable accuracy. This is notably simpler than the approach used by Zhang et al. [2], which employed ten separate scaling factors split by function type and gold content, and the results suggest that much of the accuracy can be kept with far less complexity. Package-class-specific scaling reduced the overall holdout MAE from 2.857 to 2.713 g CO_2e and represents the most reasonable lightweight refinement when additional accuracy is needed, while function-type-specific scaling produced only marginal overall gains. Whether the improvement from grouped scaling justifies the added model complexity is ultimately a practical decision to make based on priorities around simplicity and maintainability.

7.5 The CO_2e Calculator Tool

The findings from the extraction pipeline, metal prediction models, and scaling factor were integrated into a desktop application built using the **Tkinter** framework. The tool allows electrical cost engineers to enter a component's manufacturer part number and automatically retrieve its physical parameters and material composition, either from the collected dataset or by querying the manufacturer's website directly. If material content data cannot be found, the CatBoost model predicts hotspot metal weights from the available physical input parameters, and the total CO_2e emissions are computed and displayed as a sum of raw-material and process emissions. All fields can also be edited manually, making the tool usable as a direct calculation interface. The application reduces the manual effort previously required to arrive at a carbon estimate and provides a consistent, reproducible output that can serve as a second opinion in the cost engineering workflow.

7.6 Future Work

Several areas remain open for future work. Extending the data collection pipeline to cover a broader range of manufacturers would improve both the generalisability of the extraction system and the representativeness of the training dataset. As smaller open-source language models continue to improve, replacing gpt-oss 120B with a locally deployable alternative would remove the current dependency on API access and better satisfy the original goal of running the pipeline on a standard office laptop. On the modelling side, expanding the dataset and incorporating additional component types beyond the semiconductor ICs considered here could improve the robustness of both the metal prediction models and the scaling factor estimates. Finally, if manufacturers begin to disclose component-level emission figures more consistently, it would become possible to validate the end-to-end CO_2e estimates more easily, which remains the most significant gap in the current evaluation.

References

[1] Volvo Cars. *Climate Action*. URL: https://www.volvocars.com/intl/sustainability/climate-action/ (visited on 04/08/2026).

[2] Haoqun Zhang and Jinze Li. *Carbon Footprint Model for Semiconductors: Raw Material Carbon Emissions*. Degree Project in Nanotechnology. Date: 2024-08-18. Host company: Volvo Cars AB. Stockholm, Sweden, Aug. 2024. URL: https://kth.diva-portal.org/smash/get/diva2:1909096/FULLTEXT01.pdf.

[3] Vasileios Ntinopoulos et al. "Large language models for data extraction from unstructured and semi-structured electronic health records: a multiple model performance evaluation". In: *BMJ Health & Care Informatics* (2025), e101139. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11751965/.

[4] Qiyan Deng et al. "Unstructured Data Analysis using LLMs: A Comprehensive Benchmark". In: *arXiv preprint arXiv:2510.27119* (2025). URL: https://arxiv.org/abs/2510.27119.

[5] Filip Seitl et al. *Assessing the quality of information extraction*. Apr. 2024. URL: https://www.researchgate.net/publication/379661637_Assessing_the_quality_of_information_extraction.

[6] Noah Hollmann et al. "TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second". In: *Workshop on Table Representation Learning, Advances in Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://openreview.net/forum?id=eu9fVjVasr4.

[7] Noah Hollmann et al. "Accurate predictions on small data with a tabular foundation model". In: *Nature* 637.8045 (2025), pp. 319–326. URL: https://www.nature.com/articles/s41586-024-08328-6.

[8] imec. *The road to net-zero emissions in IC manufacturing*. https://www.imec-int.com/en/articles/road-net-zero-emissions-ic-manufacturing. Accessed: 2026-04-13.

[9] Zhihan Zhang et al. "DeltaLCA: comparative life-cycle assessment for electronics design". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8.1 (2024), pp. 1–29.

[10] Cheng-Hung Kuo et al. "Life cycle impact assessment of semiconductor packaging technologies with emphasis on ball grid array". In: *Journal of Cleaner Production* 276 (2020), p. 124301. DOI: 10.1016/j.jclepro.2020.124301. URL: https://www.sciencedirect.com/science/article/abs/pii/S0959652620343468.

[11] AACE International. *What Are Cost Engineering & Total Cost Management?* Accessed: 2026-02-22. URL: https://web.aacei.org/about/about-aace/what-is-cost-engineering.

[12] McKinsey & Company. *What is machine learning?* Accessed: 2026-02-22. 2024. URL: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-machine-learning.

[13] Oracle. *IQR (Interquartile Range)*. https://docs.oracle.com/en/cloud/saas/tax-reporting-cloud/ustrc/insights_metrics_IQR.html. Accessed: 2026-04-20.

[14] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[15] Joshua Noble. *What is LLM Temperature?* IBM Think. Accessed: 2026-04-13. 2024. URL: https://www.ibm.com/think/topics/llm-temperature.

[16] Shashank Verma and Neal Vaidya. *Mastering LLM Techniques: Inference Optimization*. NVIDIA Technical Blog. Accessed: 2026-04-12. Nov. 2023. URL: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/.

[17] Zixuan Zhou et al. *A Survey on Efficient Inference for Large Language Models*. Accessed: 2026-04-12. 2024. arXiv: 2404.14294 [cs.CL]. URL: https://arxiv.org/abs/2404.14294.

[18] Yi-Dong Chen et al. “A Survey of Quantization in LLM: Unlocking Potential Hardware Efficiency”. In: *Journal of Computer Science and Technology* (2026). Accessed: 2026-04-12. DOI: 10.1007/s11390-026-5979-1. URL: https://jcst.ict.ac.cn/article/doi/10.1007/s11390-026-5979-1.

[19] Erik Johannes Husom et al. “Sustainable LLM Inference for Edge AI: Evaluating Quantized LLMs for Energy Efficiency, Output Accuracy, and Inference Latency”. In: *ACM Transactions on Internet of Things* 6.4 (Nov. 2025). Accessed: 2026-04-12. DOI: 10.1145/3767742. URL: https://dl.acm.org/doi/10.1145/3767742.

[20] César Miguelañez. *How JSON Schema Works for LLM Data*. Accessed: 2026-02-27. 2023. URL: https://latitude.so/blog/how-json-schema-works-for-llm-data.

[21] Pydantic Contributors. *Why Pydantic? — Serialization*. Accessed: 2026-02-27. 2024. URL: https://docs.pydantic.dev/latest/why/#serialization.

[22] Cadence PCB Solutions. *Skills for the Coming Market: Reading Datasheets*. Accessed: 2026-02-24. URL: https://resources.pcb.cadence.com/blog/skills-for-the-coming-market-reading-datasheets.

[23] Jacob Murel and Eda Kavlakogu. *What is Ensemble Learning?* URL: https://www.ibm.com/think/topics/ensemble-learning (visited on 04/08/2026).

[24] Wikipedia contributors. *Gradient boosting — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Gradient_boosting (visited on 04/08/2026).

[25] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. New York, NY, USA: ACM, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[26] Liudmila Prokhorenkova et al. “CatBoost: Unbiased Boosting with Categorical Features”. In: *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. 2018, pp. 6638–6648.

[27] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[28] Lloyd S. Shapley. "A Value for n-Person Games". In: *Contributions to the Theory of Games*. Vol. 2. Princeton University Press, 1953, pp. 307–317.

[29] Hugh Chen. *SHAP Part 3: Tree SHAP*. Medium, Analytics Vidhya. Accessed: 2026-04-22. 2021. URL: https://medium.com/analytics-vidhya/shap-part-3-tree-shap-3af9bcd7cd9b.

[30] Simon Willison. *Structured data extraction from unstructured content using LLM schemas*. Simon Willison's Newsletter, accessed 2026-03-17. Feb. 2025. URL: https://simonw.substack.com/p/structured-data-extraction-from-unstructured.

[31] Google DeepMind. *Gemma 3 4B Instruction-Tuned Model*. https://huggingface.co/google/gemma-3-4b-it. Hugging Face model repository. 2025.

[32] Meta. *Llama 3.1 8B Instruct*. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. Hugging Face model repository. 2024.

[33] OpenAI. *gpt-oss-120b*. https://huggingface.co/openai/gpt-oss-120b. Hugging Face model repository. 2025.

[34] Prior Labs. *TabPFN: Foundation Model for Tabular Data*. 2025. URL: https://github.com/PriorLabs/TabPFN (visited on 03/30/2026).

[35] XGBoost Contributors. *XGBoost: Scalable Tree Boosting System*. 2014. URL: https://xgboost.ai/about (visited on 03/30/2026).

[36] Yandex. *CatBoost*. 2017. URL: https://catboost.ai/ (visited on 03/30/2026).

[37] SHAP Contributors. *SHAP Documentation*. https://shap.readthedocs.io/en/latest/. Accessed: 2026-03-31. 2025.

[38] Python Software Foundation. *tkinter — Python Interface to Tcl/Tk*. https://docs.python.org/3/library/tkinter.html. Part of the Python Standard Library. Accessed: 2024. 2024.

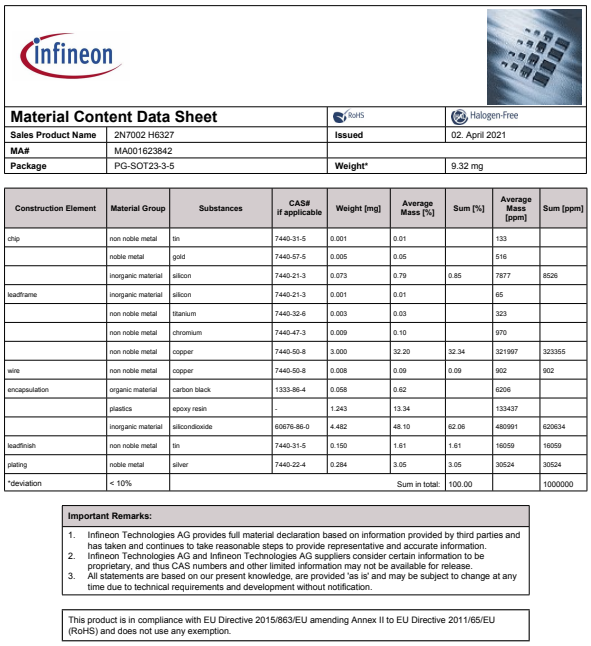
[39] Google DeepMind. *Gemma 3 QAT Models: Bringing State-of-the-Art AI to Consumer GPUs*. Google Developers Blog. Accessed: 2026-04-12. Apr. 2025. URL: https://developers.googleblog.com/en/gemma-3-quantized-aware-trained-state-of-the-art-ai-to-consumer-gpus/.

[40] LocalLLM. *Ollama VRAM Requirements: Complete 2026 Guide to GPU Memory for Local LLMs*. LocalLLM.in. Accessed: 2026-04-12. 2026. URL: https://localllm.in/blog/ollama-vram-requirements-for-local-llms.

[41] AMD. *How to Run OpenAI's GPT-OSS 20B and 120B Models on AMD Ryzen AI Processors and Radeon Graphics Cards*. AMD Official Blog. Accessed: 2026-04-12. Aug. 2025. URL: https://www.amd.com/en/blogs/2025/how-to-run-openai-gpt-oss-20b-120b-models-on-amd-ryzen-ai-radeon.html.

Appendix A

Example Material Content Sheet



Material Content Data Sheet

Sales Product Name	2N7002 H6327	**Issued**	02. April 2021
MA#	MA001623842		
Package	PG-SOT23-3-5	**Weight***	9.32 mg

Construction Element	Material Group	Substances	CAS# if applicable	Weight [mg]	Average Mass [%]	Sum [%]	Average Mass [ppm]	Sum [ppm]
chip	non noble metal	tin	7440-31-5	0.001	0.01		133	
	noble metal	gold	7440-57-5	0.005	0.05		516	
	inorganic material	silicon	7440-21-3	0.073	0.79	0.85	7877	8526
leadframe	inorganic material	silicon	7440-21-3	0.001	0.01		65	
	non noble metal	titanium	7440-32-6	0.003	0.03		323	
	non noble metal	chromium	7440-47-3	0.009	0.10		970	
	non noble metal	copper	7440-50-8	3.000	32.20	32.34	321997	323355
wire	non noble metal	copper	7440-50-8	0.008	0.09	0.09	902	902
encapsulation	organic material	carbon black	1333-86-4	0.058	0.62		6206	
	plastics	epoxy resin	-	1.243	13.34		133437	
	inorganic material	silicondioxide	60676-86-0	4.482	48.10	62.06	480991	620634
leadfinish	non noble metal	tin	7440-31-5	0.150	1.61	1.61	16059	16059
plating	noble metal	silver	7440-22-4	0.284	3.05	3.05	30524	30524
*deviation	< 10%				Sum in total:	100.00		1000000

Important Remarks:

1. Infineon Technologies AG provides full material declaration based on information provided by third parties and has taken and continues to take reasonable steps to provide representative and accurate information.
2. Infineon Technologies AG and Infineon Technologies AG suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.
3. All statements are based on our present knowledge, are provided 'as is' and may be subject to change at any time due to technical requirements and development without notification.

This product is in compliance with EU Directive 2015/863/EU amending Annex II to EU Directive 2011/65/EU (RoHS) and does not use any exemption.

Company	Infineon Technologies AG
Address	81726 Neubiberg
Internet	www.infineon.com

MCDS ID: 63958

Figure A.1: An example of a material content sheet from Infineon Technologies.

Appendix B
Screenshot of the App GUI

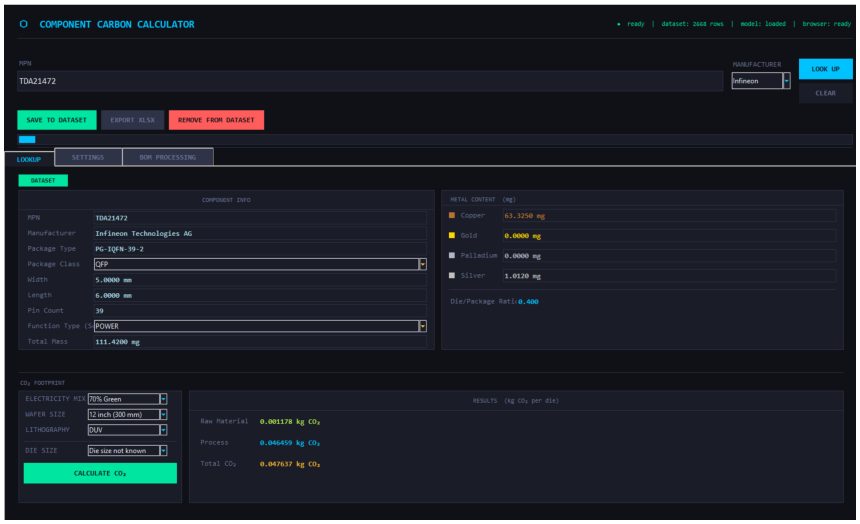


Figure B.1: A screenshot of the CO_2e tool GUI.

Printed by Tryckeriet i E-huset, Lund 2026

LUND
UNIVERSITY

Series of Master´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2026-1160
http://www.eit.lth.se