University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Szymon Kozłowski
Student no. 448304

Gustaw Blachowski
Student no. 448194

Kamil Dybek
Student no. 448224

Natalia Junkiert
Student no. 448267

Using machine learning models for processing the data presented to the user by mobile devices.

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
Jacek Sroka PhD
Institute of Informatics

Warsaw, May 5, 2025

Abstract

In the era of rapidly evolving digital applications, traditional scraping techniques face increasing challenges in maintaining reliable data collection pipelines. Commissioned by Murmuras, a company specializing in commercial and scientific data analysis, in this project we present a novel approach to processing phone screen content, such as displayed social media posts and website advertisements. Our solution leverages Large Language Models (LLMs) running locally on the user's device to handle diverse data formats while ensuring that sensitive information remains protected. The primary application explored in this study is the extraction of discount coupons, demonstrating the feasibility of our method in identifying and structuring valuable content from varying digital sources. Furthermore, the system is designed to be easily adaptable to other use cases, such as analyzing users' political views. Additionally, we explore the usage of non-LLM models for the defined task. The results highlight the potential of LLM-driven content analysis as an alternative to conventional scraping techniques.

Keywords

LLM, NLP, BERT, Android, Edge-device, Fine-Tuning

Thesis domain (Socrates-Erasmus subject area codes)

11.4 Artificial Intelligence

Subject classification

I.2.7: Natural Language Processing
H.3.3: Information Search and Retrieval

Tytuł pracy w języku polskim

Wykorzystanie modeli uczenia maszynowego do przetwarzania danych zaprezentowanych użytkownikowi przez urządzenie mobilne.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1. Project background and motivation

With the rapid advancement of information technology, the Internet has become one of the most crucial facets for many businesses to perform marketing activities [7]. One of the key marketing tools in business-to-consumer (B2C) e-commerce is the digital coupon [10]. Compared to paper coupons, digital coupons are characterized by their wide reach, rapid distribution, and low spread costs. Furthermore, a key advantage of digital coupons is their ability to facilitate targeted marketing by offering personalized discounts to different customers, thereby increasing sales [7].

Recent statistics underscore the significance of mobile devices in the domain of coupon distribution. For example, studies have shown that over 90% of digital coupon users access their vouchers via smartphones [13], and similar figures are reported by other industry sources [14]. This high rate of mobile usage creates a pressing need for coupon analysis tools that are optimized for mobile platforms, ensuring that consumers receive timely and personalized offers regardless of their location or device.

Large Language Models (LLMs) have become a fundamental technique in contemporary machine learning, replacing previously utilized recurrent neural network (RNN) architectures in the field of natural language processing (NLP) [8]. Subsequent research has demonstrated their applicability to structured input data [9], such as screen views and coupons. Additionally, there have been efforts to integrate these models into web scraping pipelines [18].

In light of these trends, the company Murmuras has tasked us with developing a solution based on a machine learning model that can be deployed as a mobile application. This model will process input representing the user's onscreen view and extract digital coupons along with their relevant data. This solution must be capable of running locally on the device, ensuring efficient processing without relying on external servers. By leveraging advanced machine learning techniques, the app will handle the diverse formats and layouts of digital coupons, thus facilitating the collection of data regarding coupons.

1.2. The definition of a coupon

A coupon is a physical piece of paper or digital voucher that can be redeemed for a financial discount when purchasing a product [3]. A coupon is characterized by a name, expiration date, and a discount type, e.g. '20% off', 'buy 1 get 1 free', etc., however, not every coupon contains each of these features. Furthermore, coupons may contain numerous other features such as images and eligibility requirements. Henceforth, the term 'coupon' will refer exclusively to a

digital coupon. The term 'conventional coupon' will refer to the traditional physical coupon. Examples of digital coupons encountered in mobile applications are presented in 1.3

Figure 1.1: Example coupon from fast-food restaurants chain mobile application

Figure 1.2: Example coupon from grocery store mobile application

Figure 1.3: Example digital coupons

1.2.1. Our data model of digital coupon

In the following research we model a digital coupon as a collection of named fields:

1. *product_name*: the name of the product,
2. *valid_until*: the text representing the date of coupon expiration,
3. *discount_text*: the text representing the discount offered to the user,
4. *activated*: either true or false, indicates whether the coupon has been activated,

We allow for special *null* value in the above fields in case no data is available.

An example of a digital coupon represented in JSON format is shown in listing 1.1:

Listing 1.1: Example of a digital coupon in JSON format

```
{
    "product_name": "Shampoo X",
    "valid_unitl": "2025-06-30",
    "discount_text": "20% OFF",
    "activated": true
}
```

1.3. The Significance of the Digital Coupon

The digital coupon is one of the most important tool in contemporary marketing strategies [10], therefore analyzing their lifecycle is essential to maximize their benefits. To facilitate such analyses, researchers collect various statistical metrics, including the fraction of redeemed coupons among all distributed coupons referred henceforth as redemption rate [5] and customer engagement [6], while also assessing their impact on sales performance [6]. Additionally, studying competitors' digital coupon strategies enables businesses to identify market trends,

adjust their promotional tactics, and maintain a competitive edge in the evolving digital marketplace.

Redemption Rate

The measurement of coupon redemption rates is primarily based on either survey data [4] or controlled experimental studies [5]. However, the company Murmuras [1] has introduced an alternative approach that enables the direct collection of coupon-related data from users' devices. This method utilizes a screen content scraping tool installed on the devices. Additionally, the tool has the ability to record user's actions. Having access to all the user's interactions and visual changes in the layout, it is possible to detect the coupon redemption. This allows for large-scale data acquisition while reducing the costs associated with traditional survey-based methods.

Customer Engagement and Impact on Sales

Customer engagement metrics, such as conversion rates and the effect of e-coupon issuance on sales, can potentially be measured using statistical analysis tools operating on the seller's website [2]. The conversion rate is typically derived by tracking visitor activity, while the impact on sales is estimated by correlating the updated conversion rate with the frequency of coupon issuance.

Although this approach provides valuable insights, it relies on direct collaboration with the coupon issuer and is constrained to a single webpage. Consequently, it is not applicable to our study, as we aim to analyze arbitrary mobile applications with diverse coupon designs.

1.4. Problem Statement

The objective of this work is to extract coupons visible to the user from the content displayed on a mobile device screen. The extracted coupons should be represented as a JSON list, with each entry conforming to the format specified in Section 1.2.1.

The screen content is provided in the form of a `.CSV` file, which encodes an XML tree structure representing the underlying screen layout. Each row in this file corresponds to a single view element within the screen hierarchy [15]. The dataset includes at least the following attributes:

1. **view_depth**: The depth of the view within the XML tree hierarchy.
2. **text**: The textual content displayed to the user within the view.
3. **id**: A unique identifier for a screen sample. Each sample consists of a set of views observed either simultaneously or in directly consecutive snapshots.
4. **time**: The timestamp indicating when the view was recorded.
5. **view_id**: The unique identifier assigned to the view by the Android API.

An example of the dataset to illustrate described format is provided in Table 1.1.

Additional requirement is that the screen content processing will be performed exclusively on the end device to mitigate potential privacy concerns.

view_depth	text	id	time	view_id
2	"50% OFF"	101	12:30:15	com.example.app:id/discount_label
3	"Buy 1 Get 1 Free"	101	12:30:15	com.example.app:id/promo_banner
2	"Limited Offer"	102	12:31:05	com.example.app:id/offer_text

Table 1.1: Example of dataset format representing screen content.

1.5. Project goals

The system will leverage machine learning to identify and structure relevant information while ensuring compatibility with mobile devices for enhanced accessibility and data privacy. The key goals of the project are as follows:

1. A tool to process the data extracted from the device into a format suitable for use by the model.
2. A machine learning tool for extracting the data that is of interest to us, such as the coupon name, expiration dates, prices, etc. The model should be capable of handling various coupon formats and layouts with high accuracy.
3. An optional tool for post-processing the output data from the tool mentioned in the previous point into a common format.
4. An application that runs the above three tools on a mobile device. (Optional)
5. A key requirement is that the machine learning model must be deployable on the mobile device itself to guarantee data privacy.

1.6. Potential applications of the project

1.6.1. Assessing coupon effectiveness

The access to the content of mobile device screen allows us to list all the coupons seen by the user. Additionally, as we will retrieve information about coupon activation status, there will be possibility to track coupon redemptions by comparing the coupons models *active* field.

Given that, our solution will aid businesses in analyzing consumer behavior and optimizing their marketing strategies. By facilitating the collection of data on coupon characteristics and their redemption rates, businesses will be able to assess the effectiveness of their coupon campaigns—determining whether they achieve the desired results. Additionally, large-scale analysis of coupon data can reveal valuable insights into purchasing patterns, preferred discount types, and the most appealing products or services.

1.6.2. Market analysis and competitor monitoring

Machine learning is proven to be a useful tool in the field of market competitors analysis but it requires significant amounts of data[11]. The aforementioned gathering of data about displayed coupons can also be utilized in further monitoring of competitors' coupon strategies, their effectiveness, and whether they provide better discounts. Using machine learning to identify and analyze competitors' strategies is more cost-effective compared to exhaustive web scraping or mystery shopping [11]. This will enable businesses to make better informed

decisions about their own marketing campaigns and provide a comprehensive understanding of the competitive landscape.

Chapter 2

Machine learning and the dangers associated with it

Over the past several years, artificial intelligence (AI) has been widely discussed in the media. Amid the promises of a utopian future, with self-driving cars and intelligent virtual assistants that dominate the headlines, concerns about AI are also growing. Many fear a future in which human labour has been made obsolete by automation and AI [29]. Privacy concerns are also mounting, as AI models are often trained on vast datasets that may include sensitive information such as healthcare records, biometric data for facial recognition, and financial details — sometimes collected without consent [36].

2.1. Understanding the difference artificial intelligence, machine learning and deep learning

Artificial intelligence (AI), machine learning (ML) and deep learning (DL) are terms often mistakenly used interchangeably to refer to the development of systems capable of performing tasks typically requiring human intelligence such as decision making and speech recognition [35]. AI is the umbrella term encompassing among others, machine learning and deep learning, as well as other approaches [30]. Machine learning is a subset of AI, in which systems are able to learn and adapt without explicit rules [35]. Deep learning is a type of machine learning utilizing neural networks. This hierarchy is depicted in the image below.

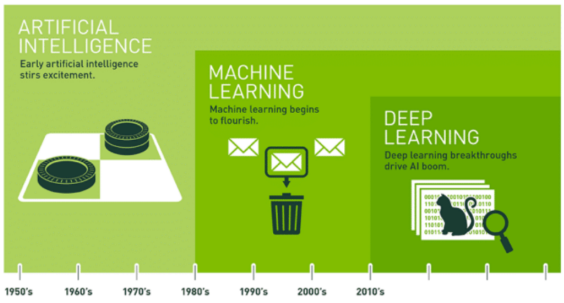

Figure 2.1: The hierarchy of artificial intelligence, machine learning and deep learning [34]

2.1.1. Artificial Intelligence

Artificial intelligence can be succinctly described as "the effort to automate intellectual tasks normally performed by humans." The field of AI encompasses various approaches, including machine learning and deep learning, which rely on data and statistical models to identify patterns and make decisions. However, the field also includes symbolic AI, which operates differently by relying solely on predefined rules rather than data-driven models [30]. An example of symbolic AI is expert systems like MYCIN, which used around 500 of IF-THEN rules to diagnose bacterial infections and recommend treatments with accuracy on par with human specialists and better than general practitioners [37].

2.1.2. Machine Learning

On the other hand, machine learning is a branch of artificial intelligence in which the machine is trained rather than explicitly programmed, by making inferences from the input data it is presented with [31]. Two major types of machine learning are supervised and unsupervised learning.

Supervised learning refers to an approach that relies on labeled datasets to train or "supervise" the model. The model is provided with input data along with the correct output, allowing it to learn by example. The model analyzes the relationship between the input features and the ground truth, gradually improving its ability to make accurate predictions. For example, to train a model to extract relevant coupon information such as the product name, new price, discount type, etc, the model is provided with coupon text and the corresponding labels such as for this example:

```
UltraComfort Ergonomic Chair - Was $199.99, Now $149.99 (25% OFF) -
Limited-time offer, free shipping available, use code SAVE25NOW at
checkout, offer expires April 10th, 2025.
```

```
{
'product_name': 'UltraComfort Ergonomic Chair',
'discount': '25%',
'old_price': '$199.99',
'new_price': '$149.99',
'other_discounts': [],
'validity': 'April 10th, 2025'
}
```

Supervised learning is commonly used for tasks such as classification, where data is sorted into categories, and regression, where numerical values are predicted based on patterns in the data. This method is widely applied in real-world scenarios like email spam detection, image recognition, and sales forecasting.

In contrast, unsupervised learning involves the model analyzing, clustering unlabeled data and identifying patterns without guidance from the programmer or datasets, hence it is called unsupervised learning. Unsupervised learning can be used to identify groups of products often purchased together [38].

In this project, we primarily employ supervised learning to develop our solution. This choice is driven by the nature of the problem, which involves identifying coupons from a screen view presented as an XML tree and extracting relevant data from them. Since this task is essentially a classification problem — where the goal is to categorize elements rather

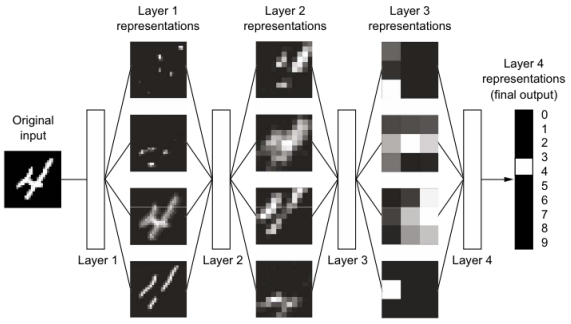

Figure 2.2: Data representations learned by a digit-classification model [32]

than uncover hidden patterns or group data points — supervised learning is the most suitable approach. By leveraging labeled data, the model can effectively learn to recognize and extract the desired information with accuracy and efficiency.

2.1.3. Deep Learning

Deep learning is a subset of machine learning wherein multilayered neural networks, called deep neural networks, are utilized to learn increasingly meaningful representations with each successive layer as seen in 2.2; each representation is increasingly different from the original and more useful to determining the result. Traditionally, while machine learning models focus on learning one or two layers of representations of the data [32], deep learning employs at least three layers, and typically hundreds or thousands of layers to train the models [56].

The transformation implemented by a layer is defined (parametrized) by its *weights*, which are numerical parameters. Learning involves adjusting these weights to ensure the network accurately maps inputs to their corresponding targets. A deep neural network can have millions of parameters, leading to complex interdependencies, since changing one parameter affects the others. To guide this process, a *loss function* measures how far the network's predictions deviate from the expected results, providing a score that reflects its performance. Deep learning relies on using the loss score as feedback to adjust the network's weights, guided by the optimizer using the backpropagation algorithm. Initially, the weights are typically random, resulting in poor predictions and a high loss score. With each example, the optimizer tweaks the weights to reduce the loss. Repeating this process across many examples gradually minimizes the loss, producing a trained network that closely matches its target outputs. This process is exemplified in 2.3 [33].

The advancement of deep learning contributed to the development of generative AI such as ChatGPT as well as Natural Language Processing (NLP) which enables machines understand and generate text and speech. This is useful for translations and extracting meaning from large quantities of data [56].

2.1.4. Transformers

Transformers are deep learning models introduced in the 2017 paper "Attention Is All You Need" [77] by Vaswani et al., which have significantly impacted natural language processing and other sequential data tasks. Unlike traditional recurrent neural networks (RNNs), transformers utilize self-attention mechanisms to process input data in parallel, enhancing

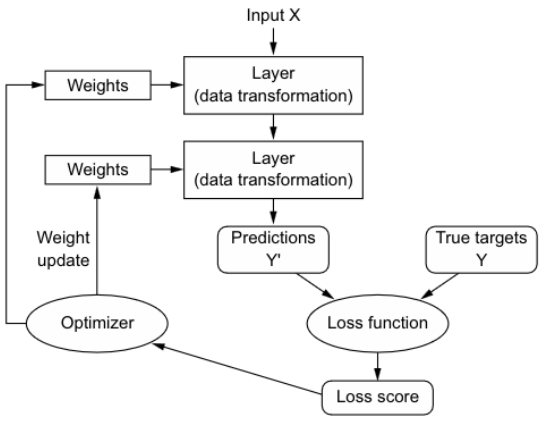

Figure 2.3: A visualization of how a Deep Learning model works [33]

efficiency and scalability. This architecture has become foundational in models like BERT, and GPT [79].

Model architecture

The proposed architecture has a encoder-decoder structure wherein, given a sequence of input symbols $(x_1, \ldots, x_n)$, the encoder computes a sequence of continuous vector representations $z = (z_1, \ldots, z_n)$, where each encodes contextual information about the corresponding within the entire input sequence. The decoder takes z and outputs a sequence $(y_1, \ldots, y_n)$. At every step, the model utilizes previously-generated symbols as additional input when creating the next output. This is shown in figure 2.4.

The encoder consists of multiple layers, each typically comprising of two parts: a multi-head self-attention mechanism and a simple fully connected feed-forward network. Residual connection and layer normalization is utilized to improve model performance.

Similarly, the decoder has a similar structure. In addition to the two parts in the encoder, each layer performs attention over the encoder's output. To ensure the model does not look ahead in the sequence, the self-attention in the decoder is modified to prevent the model from attending to future positions, so each prediction only depends on earlier information.

Please refer to the aforementioned paper for more details.

2.1.5. Quantization

Quantization is a technique employed in many fields including machine learning, to reduce the precision of numerical representations within models, typically converting high-precision formats like 32-bit floating point (FP32) to lower-precision formats such as 8-bit integers (INT8). Using integer operations instead of floating-point reduces the computational and memory requirements during inference, thereby making it more efficient and faster, which is an essential benefit for real-time applications. Furthermore, quantization enables deployment on resource-constrained hardware such as smartphones and tablets, while also reducing power consumption due to lighter computational loads. Though this may slightly reduce model accuracy, the trade-off often proves worthwhile in practical applications [74]. The two most common quantization techniques are float32 -> float16 and float32 -> int8 [75].

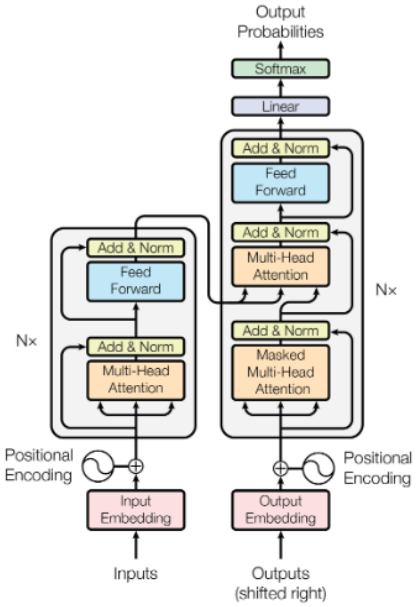

Figure 2.4: Transformer Architecture [77]

float32 -> float16

Performing quantization from float32 to float16 is relatively straightforward as both data types are represented in the same manner. Float32 has the following format:

1. The most significant bit (MBS) represents the sign of the number, ie. whether it is negative or positive.
2. The next 8 bits represent the exponent.
3. The remaining 23 bits, are the mantissa, ie. the decimal.

When converting from fromt32 to float16, we just remove the last 13 bits of the mantissa, creating a rounding error, and "shrinks" the exponent to fit into 5 bits. This may create a float overflow error if the float32 number is greater than $6.55e^4$ [76].

float32 -> int8

This type of quantization is more complicated as an int8 can only represent 256 values which is significantly less compared to the 2^24 values represented by a float32. The idea of this quantization is to find the best way to map float32 values into the int8 space, using the affine quantization scheme: $x = S * (x_q - Z)$.

1. x is the float32 value to be quantized.
2. x_q is the quantized int8 value associated with x. It can be computed as follows $x_q = round(x/S + Z)$.
3. $S \in float32$ is the scale.
4. Z is the zero-point, which is the int8 value that corresponds to 0 in the float32 range [75].

2.1.6. Finetuning

Fine-tuning is a specialized form of transfer learning that involves adapting a pre-trained model to perform a specific task, such as identifying coupons or extracting relevant information from them, rather than identifying all kinds of objects. Instead of training a model from scratch, fine-tuning starts with a model already trained on a large dataset and further trains it using a smaller, task-specific dataset. Attempting to train a large model from scratch on a small dataset can lead to overfitting, where the model performs well on training data but generalizes poorly to unseen data. This approach is particularly beneficial for deep learning models, such as large language models (LLMs) in natural language processing or convolutional neural networks (CNNs) in computer vision, as it reduces the computational resources and labeled data required [72]. Fine-tuning consists of the following steps:

1. Training a source model on a large, general-purpose dataset. This enables the model to learn broadly useful feature representations.

2. Construct a new target model by copying the architecture and parameters of the source model, excluding its final output layer. The retained parameters are presumed to encode transferable knowledge, while the output layer—being specific to the source dataset —is discarded.

3. Introduce a new output layer tailored to the target task, ensuring it matches the number of classes in the target dataset. Initialize this layer's parameters randomly.

4. Train the target model on the new, task-specific dataset (e.g., a dataset of coupons. The new output layer is trained from scratch, while the remaining layers are fine-tuned using the pre-trained weights as a starting point [73].

2.2. Should We Be Afraid of AI? - Assessing the Risks and Ethical Implications of Artificial Intelligence

As artificial intelligence becomes increasingly complex and integrated into our daily lives, the voices raising concerns about its dangers grow louder. Some express concern about the risks of excessive surveillance and privacy erosion, envisioning a future where AI systems are deeply intertwined with our surroundings, essentially hearing what we hear and seeing what we see [40] [41] [42]. Others highlight the environmental challenges tied to large-scale AI deployment, such as rising energy and water consumption, along with the need for rare materials in microchip production [43]. This section will focus on assessing the risks of artificial intelligence and how our solution will take them into account.

2.2.1. Privacy Erosion

AI systems heavily rely on vast amounts of user data to deploy machine learning techniques that identify subtle patterns and behaviors that may not be immediately evident, thus enabling personalized recommendations [46]. Social media platforms like TikTok exemplify this, as their algorithms suggest content based on users' previous interactions [44] [45]. While this personalization enhances user engagement, it also risks influencing people's opinions and shaping their worldview by trapping them in ideological echo chambers[70].

Consequently, 'information privacy' is one of the primary concerns surrounding the use of AI. 'Information privacy' refers to the protection of personal data that is being collected,

processed and stored by AI systems [47]. Training machine learning models typically requires immense datasets, involving terabytes or even petabytes of information. Therefore, these training sets likely include sensitive user information such as healthcare records and biometric data [48]. Beyond explicit data collection, AI systems can also infer highly personal attributes, such as political beliefs, sexual orientation, or health conditions, from seemingly unrelated data — a phenomenon known as 'predictive harm' [47]. This information can be utilized to subject individuals to targeted advertising, unwanted profiling, and even identity theft, often without the user's consent or awareness [46], thus posing a serious privacy risk.

Moreover, AI systems can pose 'autonomy harms', wherein the insights derived from data are used to influence individuals' decisions or behavior without their knowledge or proper consent. The common mindset of having "nothing to hide" overlooks the broader implications of these practices, which can undermine personal freedom and informed decision-making. A significant example of this problem is the Facebook-Cambridge Analytica scandal, in which a seemingly benign personality quiz was utilized to harvest over 87 million Facebook users' data. Based on this data, detailed psychological profiles were constructed and leveraged to target individuals with personalized political ads during the 2016 US Presidential Election. This case underscores how AI can extract deeply personal insights from mundane user interactions, demonstrating the potential for misuse when data privacy protections are inadequate [47].

To address privacy concerns associated with AI systems, we are implementing locally-deployable models that process data directly on users' devices. This ensures that raw, unprocessed information never leaves the device or gets transmitted to external servers. By keeping data local, we significantly reduce the risk of unauthorized access, data breaches, or misuse. This approach empowers users with greater control over their information while still benefiting from the capabilities of AI in a secure and privacy-conscious manner.

2.2.2. Environmental Concerns

AI has a notable carbon footprint due to its increasing energy consumption, particularly during model training and usage [49]. One study predicts that by 2027, AI-related energy consumption could reach 85–134 TWh [50], representing nearly 0.5% of today's global electricity usage. This estimate is based on the energy consumption of the Nvidia A100 servers - the hardware estimated to be used by 95 percent of the A.I. market, and their projected sales in the upcoming years [51]. Training LLMs typically requires significantly more energy in comparison to making a single prediction on the trained model [52], for instance BERT required "the energy of a round-trip transcontinental flight" to train, while GPT-3 emitted 552 metric tons of carbon dioxide which is "the equivalent of 123 gasoline-powered passenger vehicles driven for one year" [53].

Furthermore, while a quarter of the world lacks access to clean water and sanitation, data centers consume significant quantities of water during their construction and during operation to cool electrical components [55]. AI server cooling requires up to 9 liters of water per kWh of energy used. Given the amount of energy required to train and operate a model, this will exacerbate the lack of water, with UN estimates stating that by 2030 half of the world's population will be facing severe water stress [54]. It is estimated that globally, infrastructure related to AI will soon require six times more water than Denmark [55].

In our approach, we opted to fine-tune existing language models rather than train new ones from scratch, a decision driven by both computational efficiency and environmental considerations. As highlighted by Wang et al. (2023), pre-training a model like BERT can require the equivalent energy of anywhere from 400 to 45,000 fine-tuning runs, depending on the dataset size. Their analysis shows that the number of training tokens is a reliable

heuristic for estimating energy use during fine-tuning, and that sequence length significantly impacts energy intensity during this phase. These findings emphasize the substantial energy and carbon costs associated with pre-training, underscoring the environmental benefits of leveraging pre-trained models for downstream tasks [71].

Chapter 3

Overview of Existing Solutions

To the best of our knowledge, at the time of this project's commissioning, no publicly available solutions directly addressed this problem. The most comparable approaches involve existing multimodal models. While widely used models such as ChatGPT and Gemini provide general data extraction capabilities [16], they are unsuitable for our task due to their substantial computational requirements. A key limitation of these models is their large size—for example, GPT-3 consists of 175 billion parameters[17]—rendering them impractical for deployment on mobile devices [27].

Alternatively, computer vision models can be used to extract text and bounding boxes from screen images. Microsoft's OmniParser [24], for instance, has demonstrated strong performance on the ScreenSpot dataset [24, 25]. However, the challenge of organizing extracted text into structured coupon data renders this approach unsuitable for our study. Furthermore, our experiments with running OmniParser locally on example images indicate that it relies on CUDA technology, making it impractical for deployment on mobile devices.

3.1. Murmuras' existing solution

Murmuras' current approach relies on a set of fixed scrapping programms tailored to specific layouts from limited set of applications, making it inflexible and expansive to generalize across diverse coupon formats. This lack of adaptability limits its usefulness in real-world scenarios where coupon structures vary widely. Since our goal is to develop a solution that is easily adaptable for processing diverse mobile content, this method is not well-suited for our needs.

In contrast, Murmuras' most-recent proof of concept involves wrapping the `CSV` data with a prompt that instructs the model and sending it to GPT-4o-mini. This approach leverages an LLM to interpret the data to extract relevant coupon details. However, the reliance on an external server means the solution does not run locally on the mobile device, leading to potential privacy concerns, latency issues, and a dependence on internet connectivity.

3.2. Scapegraph AI

ScrapeGraphAI is an open-source library that streamlines data extraction from websites and local documents by utilizing LLMs and graph logic to construct scraping pipelines [18]. The library supports integration with various LLMs, including local models through the use of Ollama [19] [20].

However, Scrapepraph AI provides only Python and Node.js SDKs [21], which could prove to be an issue with regard to mobile deployment, because neither Python nor Node.js is

natively supported on iOS or Android [22] [23].

Moreover, due to mobile devices typically having limited processing power and memory compared to desktop computers or servers [26], we cannot solely rely on the size of the model in order to improve performance. We believe that through fine-tuning LLMs, we are able to develop tools that are far more viable for edge device usage.

Chapter 4

Description of datasets

4.1. Raw datasets

We received datasets corresponding to six different mobile applications: Edeka, Rossmann, DM, Rewe, Lidl, and Penny. Each dataset was provided as a CSV file containing XML representations of the data captured from the device interface during user interactions with the respective app. The selection of these applications, as well as the data collection and processing procedures, were carried out by Murmuras.

For each application, two key files were provided: the *content_generic* CSV file, which contains screen-captured data in XML format (see 4.1), and the *coupons* CSV file, which serves as the ground truth (see 4.2). The latter includes the coupons that our solution is expected to identify, along with the specific information that should be extracted from them.

We have streamlined the tables by removing the columns that were deemed unnecessary for our project, as their inclusion would have added excessive complexity without providing relevant value.

id	time	i	view_depth	text	description
12240295656	2024-03-07T16:55:32.405	1	3	Coupon Text	

seen_timestamp	is_visible	x_1	y_1	x_2	y_2
1709826932091	true	0	81	1080	2188

Table 4.1: Content generic file format (split view for readability)

4.2. Dataset Formats

We preprocess the raw data to ensure it is structured in a manner that can be effectively utilized by the models.

4.2.1. BERT Datasets

The BERT models accept two dataset formats:

1. **Plain format:** This format consists of raw, concatenated texts extracted from the ‘text‘ field in the `content_generic` CSV file.

id	time	coupon_i	visible
12240296756	2024-03-07T16:55:51.831	2	1

discount_details	validity_text	activation_text
Vor Aktivierung bitte Hinweise lesen.	Gültig bis 31.03.2024	

content_full
"[Düfte , Vor Aktivierung bitte Hinweise lesen. , 15% , Gültig bis 31.03.2024]"

content_proper	product_text
"[Düfte]"	Düfte

extra	discount_text	product	limitation	validity_date
	15%	Düfte		2024-03-31

Table 4.2: Coupon file format (split for readability)

2. **XML Tree format:** This format encodes the data from the `content_generic` CSV file into an XML tree structure, which is then represented in JSON format, as shown in 4.1.

```
{
   "text": "text field content",
   "children": {
       "child1_view_id": ...,
       "child2_view_id": ...,
       ...
   }
}
```

Figure 4.1: Sample JSON Format for BERT Dataset

4.2.2. Llama Datasets

The Llama models take in the following dataset formats:

1. **one_input_multiple_outputs_wrequest**: the dataset includes a request to the Llama model.

2. **one_input_multiple_outputs_wthrequest**: the dataset does not include a request.

The input format for the Llama models is as in 4.2 and 4.3.

```
{
   {prompt text}
   +
   ## Input: {coupon text extracted for one timestamp}
   +
   ## Response:{response text}
}
```

Figure 4.2: Llama input format with a prompt

```
{
   ## Input: {coupon text extracted for one timestamp}
   +
   ## Response:{response text}
}
```

Figure 4.3: Llama input format without a prompt

4.2.3. Dataset statistics

This section provides an overview of the dataset statistics along with a brief analysis.

Store	Split	Bytes	Examples
Edeka	Train	632,413	298
	Test	208,987	75
DM	Train	6,863,888	2,317
	Test	1,734,466	580
Lidl	Train	8,160,797	3,180
	Test	2,194,200	796
Penny	Train	51,263	79
	Test	11,724	20
Rewe	Train	4,210,545	1,724
	Test	1,051,741	432
Rossmann	Train	3,806,354	1,476
	Test	942,792	370

Table 4.3: Dataset sizes and example counts per application and split for the coupon selection stage.

Store	Split	Examples	Bytes
Penny	Train	76	77,006
	Test	20	15,834
Lidl	Train	3,183	10,186,116
	Test	796	2,094,306
Rewe	Train	1,728	5,811,132
	Test	432	2,223,958
Edeka	Train	295	762,506
	Test	74	118,532
DM	Train	2,279	6,720,006
	Test	570	2,739,686
Rossmann	Train	1,476	3,883,376
	Test	370	1,220,512

Table 4.4: Dataset size and example counts per application and split for the Llama models

Store	Split	Examples	Bytes
Edeka	Train	52	7,683
	Test	14	1,345
DM	Train	748	106,284
	Test	187	22,945
Lidl	Train	1,773	223,428
	Test	444	55,066
Penny	Train	38	3,635
	Test	10	823
Rewe	Train	668	126,634
	Test	167	31,203
Rossmann	Train	366	57,919
	Test	92	14,303

Table 4.5: Dataset size and example counts per application and split for the BERT models

Company	Discount Text	Product Name	Valid Until	Activation Text	Missing (%)
DM	1,497	1,614	1,530	2,095	50.15%
Edeka	58	17	338	336	92.68%
Lidl	235	104	242	258	3.01%
Penny	0	0	0	36	53.73%
Rewe	45	63	20	3	0.81%
Rossmann	0	0	7	18	0.68%

Table 4.6: Missing feature counts and percentage of coupons missing at least one feature per company

Chapter 5

Technologies

5.1. General

The main platforms used during the development process were Hugging Face[84] and GitHub[85]. Hugging Face served as a hub for storing and managing both trained machine learning models and datasets. During development, we also made extensive use of several Python libraries provided by Hugging Face, including *datasets*[93], *transformers*[95], and *evaluate*[94]. GitHub, in turn, offered version control for our code and documentation, and supported DevOps tasks such as issue tracking and automated unit testing.

Python[88] was the primary programming language used, due to its broad support for machine learning workflows. Additionally, some auxiliary experiments related to Android deployment were conducted using Kotlin[90][89].

5.2. LLaMA Proposal

Our first solution was based on the LLaMA architecture and utilized a custom fine-tuned version of LLaMA-3.2-1b provided by Meta[82]. Fine-tuning was performed using the Unsloth[83] tool, which enabled efficient resource usage. Specifically, Unsloth applied the LoRA (Low-Rank Adaptation) fine-tuning technique[91], reducing the number of trainable parameters to approximately 10 million. It also facilitated easy model conversion to the GGUF format.

The fine-tuning process was carried out on the Modal platform[86], which allowed us to define execution environments directly in Python and execute code on cloud infrastructure equipped with H100 GPUs. Training progress and logs were monitored using Weights & Biases (Wandb)[87].

Model inference was conducted using the Llama.cpp[96] tool, utilizing the model converted to the GGUF format.

5.3. BERT Proposal

The second approach—a two-stage BERT pipeline—was also trained on the Modal platform. Logging and experiment tracking were similarly managed using Wandb.

5.4. Auxiliary Experiments

Additional experiments conducted as part of this project involved frameworks such as SpaCy[97][104] and Scrapegraph AI[18][105], as well as mobile deployment tools including ONNX[98][106], liteRT[99][107], and Executorch[100][108]. These experiments were developed using Android Studio[101][89] and Jupyter notebooks[102], running in an IPython[103] environment.

Chapter 6

BERT 2-stage architecture overview

Our proposed solution addresses the problem by framing it as a Named Entity Recognition (NER) task and employing a pipeline of two BERT models. Introduced by Google in 2018 [57], BERT has become a standard in NLP - as of 2025-04-04, it ranks as the fourth most downloaded model on HuggingFace [58]. The base model contains 110 million parameters, and its output format is well-suited for token classification — this combination was the primary reason for choosing it. However, we observed that the model struggled to accurately predict coupon attributes solely within the target regions when processing complex input data (see 1).

To mitigate this, we designed a two-stage architecture inspired by region proposal networks in computer vision [59]. The first stage identifies coupon regions, while the second extracts their attributes. This hierarchical approach significantly reduces false positives outside coupon boundaries.

6.1. BERT family of models

The widespread adoption of BERT [58] has led to numerous architectural variants. Beyond the larger BERT-Large (around 300M parameters) and multilingual adaptations, efforts have focused on enhancing performance (e.g., RoBERTa [60], which outperforms vanilla BERT through extended pretraining) and improving efficiency (e.g., ALBERT [61], which reduces parameters to around 12M [62]). Additionally, during our research, ModernBERT [67] was released, offering advancements such as extended context length (8,192 tokens), faster inference, and superior performance on benchmark tasks. On top of that, it was still comparable with the base model, having around 150M parameters [68]. After evaluating these variants, we selected the BERT-base multilingual model (179M parameters [66]). This decision was

```
{
    "texts": ['Wasch-', '&', 'Reinigungsmittel', 'Vor',
    'Aktivierung', 'bitte', 'Hinweise', 'lesen.',
    'Deos,', 'Duschen,', 'Badezusätze', '&', 'Bodylotions',
    'Vor', 'Aktivierung', 'bitte', 'Hinweise', 'lesen.']
}
```

Listing 1: Pipeline input

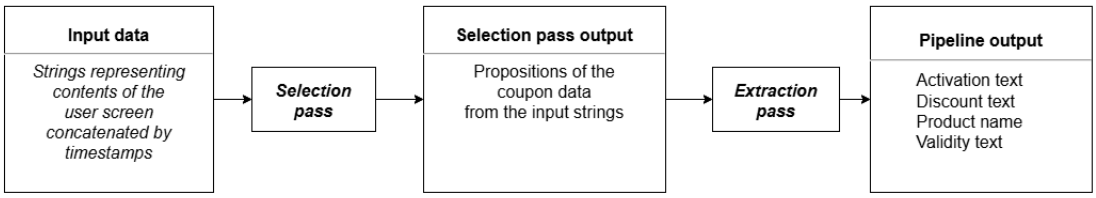

Figure 6.1: Pipeline graph

driven by ALBERT's significant performance drop [64] while both RoBERTa and ModernBert lacked multilingual support [65], making BERT-base the optimal balance of capability and versatility for our task.

6.2. 2-stage architecture

Figure 6.1 represents the general structure of the system. It consists of:

1. **Input Data** (Listing 1):
   - Each frame captured by the user's phone is stored as an XML tree
   - XML data is flattened into a CSV file where entries from the same frame share a timestamp value
   - All text data from a frame is concatenated (grouped by timestamp) and fed into the model
2. **Selection Pass**:
   - First multilingual BERT model performs Named Entity Recognition (NER)
   - Tags tokens as either `COUPON` or `UNKNOWN` in IOB2 format [81]
3. **Selection Pass Output**:
   - Labeled tokens are reassembled into contiguous strings (grouping adjacent tokens with the same label)
   - Only text segments labeled `COUPON` are forwarded to the next stage
4. **Extraction Pass**:
   - Second multilingual BERT model processes the filtered input
   - Performs fine-grained NER to classify tokens into four fields:
     - `Activation`
     - `Discount`
     - `Product`
     - `Validity text`

```
[
    {
        "ACTIVATION-TEXT" : string,
        "DISCOUNT-TEXT" : string,
        "PRODUCT-NAME" : string,
        "VALIDITY-TEXT" : string,
    }
],
...
```

Listing 2: Pipeline output

There may be cases where multiple entities are classified under the same type, such as "ACTIVATION-TEXT". To handle this, we implemented two approaches. The first approach selects only the first entity of the given type, while the second approach concatenates all entities of that type. Both methods were evaluated in benchmark tests.

5. **Pipeline Output**:
    - Tokens are concatenated into strings based on their field labels
    - Results are packaged into JSON objects
    - Final structure matches the format shown in Listing 2

6.3. Fine Tuning

We implemented independent training for both BERT models in our pipeline. This approach provides several key advantages:

- **Isolated Error Penalization**: Each model is only penalized for its own mistakes during training. Joint training could distort the selection model's loss function due to errors propagating from the extraction pass.
- **Data Integrity**: The extraction model trains on accurate intermediate representations, preventing potential bias from the selection model's imperfect predictions during joint training.
- **Practical Benefits**:
    - Simplified implementation compared to end-to-end training
    - Greater control over each model's learning process
    - Easier hyperparameter tuning for individual components

This modular training strategy ensures both models reach their optimal performance without compromising each other's learning objectives.

6.3.1. Fine-tuning methodology

Both selection and extraction models were trained on a dataset comprising data from four retail applications: DM, Lidl, Rewe, and Rossmann. The dataset was split 80%–20% for training and testing respectively, following common practice [80], with evaluation performed after each epoch. All fine-tuning experiments were conducted under the assumption that final model performance would be benchmarked on separate datasets from Edeka and Penny applications to maintain independence between training and evaluation data.

1. **General Fine-tuning**:
   - Objective: Establish baseline performance metrics
   - Approach: Models trained on combined data from all four applications
2. **Per-application Fine-tuning**:
   - Objective: Evaluate cross-application generalization capabilities
   - Approach: Individual models fine-tuned separately on each application's data, then tested on other applications to measure dataset similarity
3. **Incremental Separate Fine-tuning**:
   - Objective: Test knowledge extensibility through sequential learning
   - Approach: Four progressive models for each pass:
     (a) Base model fine-tuned only on DM data
     (b) Subsequent fine-tuning on Lidl data
     (c) Additional fine-tuning on Rewe data
     (d) Final fine-tuning on Rossmann data
4. **Incremental Combined Fine-tuning**:
   - Objective: Evaluate cumulative learning effects
   - Approach: Similar to incremental separate fine-tuning but with concatenated datasets:
     (a) Initial fine-tuning on DM data only
     (b) Subsequent fine-tuning on combined DM + Lidl data
     (c) Additional fine-tuning on DM + Lidl + Rewe data
     (d) Final fine-tuning on all four applications' data

6.3.2. Selection pass specifics

Beyond the basic fine-tuning approach, we conducted two additional experiments with the selection pass model:

1. **XML Tree Preservation**:
   - *Hypothesis*: Maintaining structural information from the original XML tree could improve coupon detection
   - *Implementation*: Created datasets where screen content strings were wrapped in JSON structures simulating the XML hierarchy
   - *Results*:

- Increased training cost due to additional tokens (e.g., structural delimiters)
- Ultimately underperformed compared to plain text approach
- Abandoned after general fine-tuning trials (see Appendix B for details)

2. **Token Distribution Balancing**:

   - *Motivation*: Early datasets contained significant class imbalance with excessive non-coupon data. Additionally, JSON format introduced even more UNKNOWN tokens.
   - *Implementation*: Applied techniques to rebalance token distribution
   - *Findings*:
     - Later data batches naturally resolved the imbalance issue
     - Final performance remained unsatisfactory (see Appendix A)
     - Approach was discontinued

Both experimental approaches were ultimately abandoned in favor of the simpler, more effective plain text processing method.

6.4. Selection pass fine-tuning results

General fine-tuning experiment demonstrated that a simple data representation without any class balancing yielded the best results. As a consequence, we decided to abandon both the JSON format and the Curriculum Learning approach. Furthermore, subsequent experiments were conducted using 10 epochs instead of 30, as higher epoch counts led to increasing loss values for the base model. In the plots (C.1, C.2, C.3), models labeled as "curr" were trained using the Curriculum Learning algorithm.

For per-application fine-tuning, models generally performed poorly achieving recall statistic on data they had not encountered during training. An exception was the Lidl dataset, where the model achieved over 60% recall (C.4).

In both incremental fine-tuning scenarios, although the model initially trained on DM data struggled with classification, further fine-tuning significantly improved its recall. This suggests that it is possible to extend the model's functionality with minimal effort through simple fine-tuning (C.5, C.6).

6.5. Extraction pass fine-tuning results

In the general fine-tuning setting, the extraction pass was learned smoothly. A potential concern, however, is the simultaneous increase in loss, recall, and precision, which may indicate slight overfitting (D.1, D.2, D.3).

The single-application experiment produced results similar to those observed in the selection pass. Once again, the Lidl dataset appeared to be the most similar to the others (D.4).

For both types of incremental experiments, the results closely mirrored those of the selection pass. However, in this case, fine-tuning on the Lidl dataset led to even greater performance improvements, making additional fine-tuning less impactful. This indicates that with the right dataset, a relatively small amount of data may be sufficient to expand the model's capabilities (D.5, D.6).

Chapter 7

LLamas

7.0.1. TODO

Chapter 8

Benchmark and Results

The objective of the benchmark is to enable a fair comparison of the pipelines with respect to the quality of their outputs. To achieve this, the benchmark compares the coupons extracted by each pipeline against the ground truth data provided by Murmuras and counts the number of correctly extracted coupons. This is accomplished by computing the similarities between the generated and expected coupons and applying a simple greedy matching algorithm.

8.1. Coupon Similarity

The coupon similarity metric employed in this work is based on comparing text fields using a string distance measure (for further details, see Appendix E). The resulting similarity score ranges from 0, indicating completely different texts, to 1, indicating identical texts. An alternative approach was considered, in which a large language model (LLM) would be used to assess the similarity between the expected and generated outputs [110]. The motivation for this consideration is that the generative pipeline, based on a model such as Llama, could produce a semantically correct answer but in an incorrect format, resulting in a low string similarity score despite the answer being essentially correct. However, we were unable to identify any documented instance where the LLM-as-a-Judge technique was successfully applied to a task sufficiently similar to ours, and providing a thorough validation of this approach falls outside the scope of this thesis.

8.2. Benchmarking Algorithm

For each entry in the benchmarking dataset, the similarities between all expected and generated coupons are computed. The matching process then proceeds iteratively: at each step, the coupon pair with the highest similarity score is selected and marked as matched. The matched coupons are removed from their respective sets, and the next highest-scoring pair is selected from the remaining coupons. This process is repeated until the highest remaining similarity score falls below a predefined threshold. In this work, the threshold was set to 0.8. The benchmark returns the total numbers of expected, generated, and matched coupons across the entire benchmarking dataset.

8.3. Logging

The benchmark logs both the results for individual entries and any unexpected events encountered during processing. For example, during the parsing of the JSON output generated

by the pipeline, any instance where a coupon lacks a product name or contains unexpected fields is recorded in the logs.

8.4. Benchmarking Results

By interpreting the number of expected coupons as the sum of true positives and false negatives, the number of generated coupons as the sum of true positives and false positives, and the number of matched coupons as the number of true positives, we can compute both recall and precision. Following discussions with Murmuras, it was determined that recall is the more important metric in our context, as correctly extracting as many coupons as possible is the primary objective, while false positives are considered to be of lesser concern.

Based on Figures F.1 and F.2, BERT-concat was identified as the best-performing variant of the BERT pipeline. Similarly, Figures F.3 and F.4 indicate that Llama-wth outperforms Llama-w. When comparing BERT-concat to Llama-wth (Figures F.5 and F.6), the latter clearly emerges as the superior approach in terms of output quality.

8.5. Comparison of Llama Quantizations

After identifying Llama-wth as the best-performing pipeline variant, we evaluated several quantized versions of the underlying model—specifically `Q8_0`, `Q4_K_M`, and `Q4_0` [111]—in terms of recall, precision, inference speed on a mobile device, and GGUF file size.

The model speed was evaluated by running the Llama-wth model with various quantization schemes using the `llama.cpp` framework [96] with the default CPU backend on a Samsung A25 device equipped with 6 GB of RAM. An initial warm-up run was performed using a short prompt of approximately 10 tokens, during which 16 tokens were generated. For the actual measurement, a prompt of around 300 tokens—sampled from one of the datasets—was used, and the model generated 64 tokens.

Based on the results shown in Figures G.1, G.2, G.3, and G.4, we found that quantization introduced no substantial degradation in output quality while offering clear improvements in inference speed and reduced model size. The latter also contributes to lower memory (RAM) usage during execution.

Chapter 9

Conclusion

9.0.1. TODO

Appendix A

Curriculum Learning Approach

To address class imbalance in Murmuras-provided data and the JSON-based selection pass, we implemented a curriculum learning algorithm inspired by curriculum learning [69]. This approach mimics human learning progression by gradually increasing task difficulty:

- **Implementation**:
  - Initial training on balanced datasets (equal coupon/non-coupon examples)
  - Progressive introduction of more non-coupon samples
  - Gradual complexity increase mirroring human learning patterns
- **Observed Advantages**:
  - Smoother, more stable loss descent during training
  - Reduced overfitting compared to baseline approaches
  - More predictable learning trajectory
- **Performance Outcomes** C.1, C.2, C.3:
  - Consistently inferior metrics (accuracy/recall) versus baseline
  - 30-epoch evaluation showed no competitive improvement
  - Final decision to exclude from production pipeline

Despite its theoretical benefits and training stability, the curriculum approach was ultimately abandoned as baseline methods demonstrated superior practical performance across all key metrics. This suggests that for our specific task and data characteristics, direct exposure to the true data distribution may be preferable to gradual introduction of complexity.

Algorithm 1 Curriculum Data Preparation Algorithm

Require: Dataset D with fields `texts`, `labels`
Require: Number of splits $S > 0$
Ensure: Sequence of training datasets

Initialize $\mathcal{R}_c \leftarrow \{\}$, $\mathcal{R}_{\neg c} \leftarrow \{\}$
for each row r in D **do**
    if r contains coupon labels **then**
        Extract spans of contiguous coupon tokens
        Add span info to $\mathcal{R}_c$
    else
        Add r to $\mathcal{R}_{\neg c}$
    end if
end for
Store init_len $\leftarrow |\mathcal{R}_c|$
for each row r in $\mathcal{R}_c$ **do**
    Extend spans of r proportionally using ExtendSpans
end for
Yield initial dataset from $\mathcal{R}_c$
for $i = 1$ to $S - 1$ **do**
    if $i = S - 1$ **then**
        Append all of $\mathcal{R}_{\neg c}$ to $\mathcal{R}_c$
    else if $i \bmod 2 = 0$ **then**
        Append a subset of $\mathcal{R}_{\neg c}$ to $\mathcal{R}_c$
    else
        for each row r in $\mathcal{R}_c$ **do**
            Extend spans of r proportionally
        end for
    end if
    Yield dataset from $\mathcal{R}_c$
end for

Algorithm 2 ExtendSpans Procedure

procedure ExtendSpans($spans, amount, max\_len$)
    if $spans = \emptyset$ **then**
        Create single span $[0, amount]$
    else
        Distribute *amount* evenly across spans
        Shift span boundaries without overlap
        Greedily expand remaining budget
    end if
end procedure

Appendix B

BERT Selection pass JSON format

For the selection pass in our two-stage architecture, we evaluated two alternative data formats:

- **Plain format**: A direct representation of the textual screen content for each timestamp (Listing 1)
- **JSON format**: An enriched representation that additionally encodes the XML tree structure of the Android screen content (Listing 3)

Following comprehensive fine-tuning experiments, we discontinued the JSON format approach due to several factors:

- Performance metrics during training were comparable to or worse than the curriculum learning approach
- The JSON structure increased token count, resulting in:
  - Longer training times
  - Higher computational costs
- The approach failed to deliver satisfactory improvements in model accuracy

The comparative results are presented in Figures C.1, C.2, and C.3.

```
        {
            "text": null,
            "children": {
                "nan": {
                    "text": "Sensodyne",
                    "children": {}
                },
                "nan_0": {
                    "text": "G\\u00fcltig', 'bis', '15.09.2024",
                    "children": {}
                },
                "nan_1": {
                    "text": "Yippy",
                    "children": {}
                }
            }
        }
```

Listing 3: Pipeline JSON format input example

Appendix C

BERT selection pass fine-tuning results

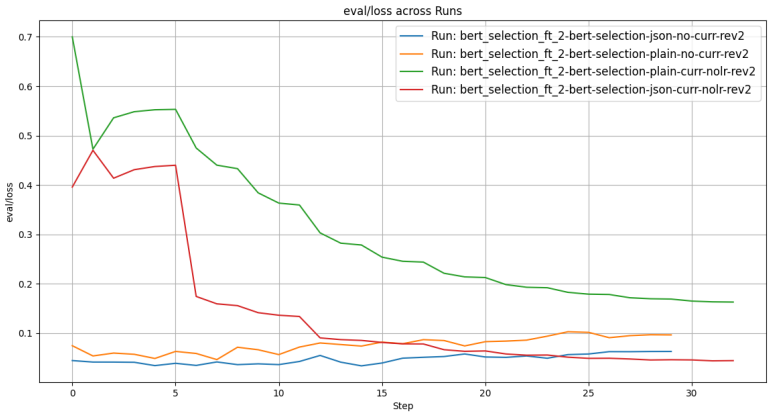

Figure C.1: Eval/loss statistics of selection pass during general training

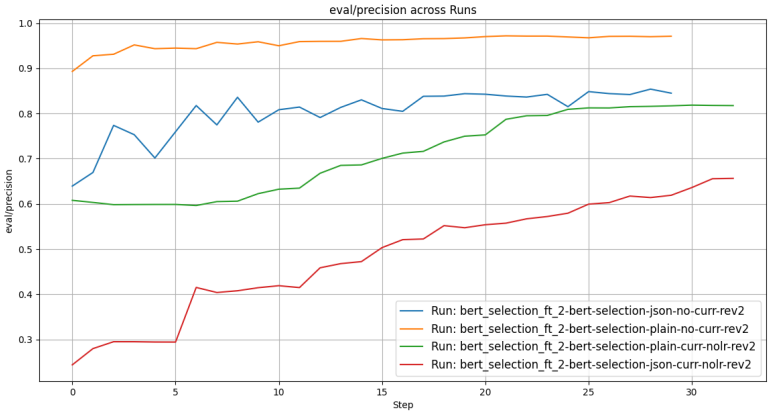

Figure C.2: Eval/precision statistics of selection pass during general training

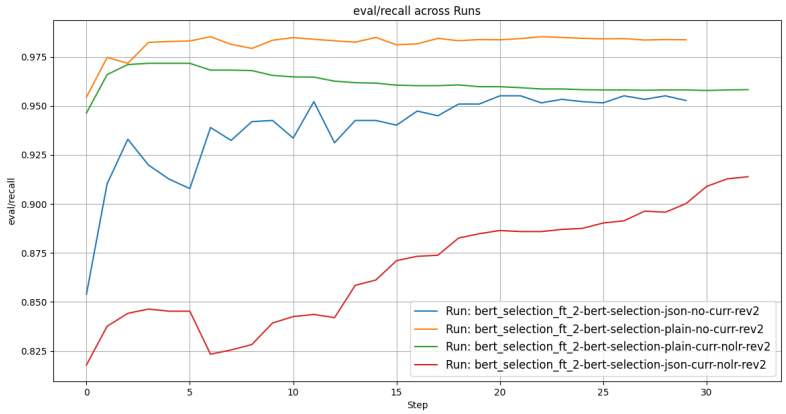

Figure C.3: Eval/recall statistics of selection pass during general training

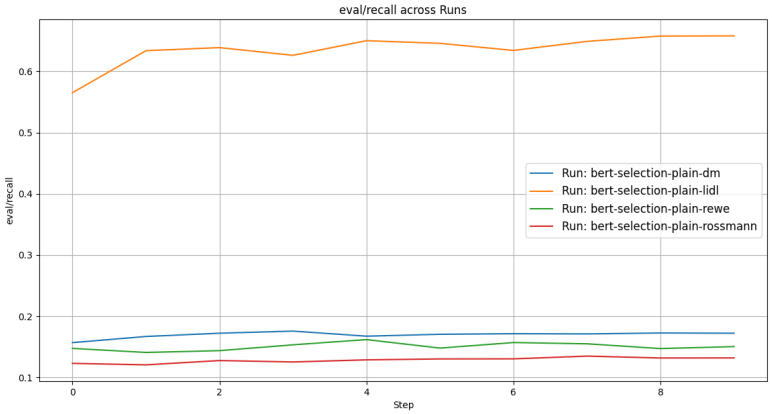

Figure C.4: Eval/recall statistics of selection pass during training on separate apps

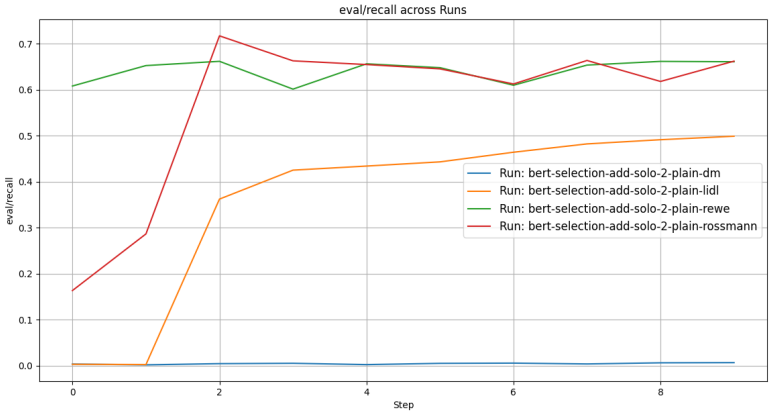

Figure C.5: Eval/recall statistics of selection pass during incremental separate training

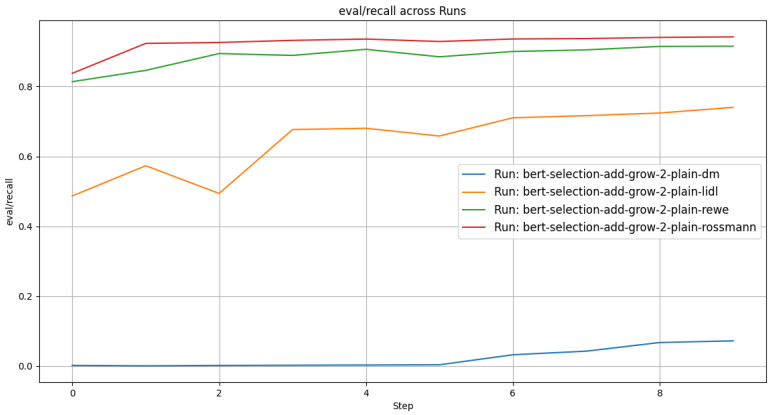

Figure C.6: Eval/recall statistics of selection pass during incremental combined training

Appendix D

BERT extraction pass fine-tuning results

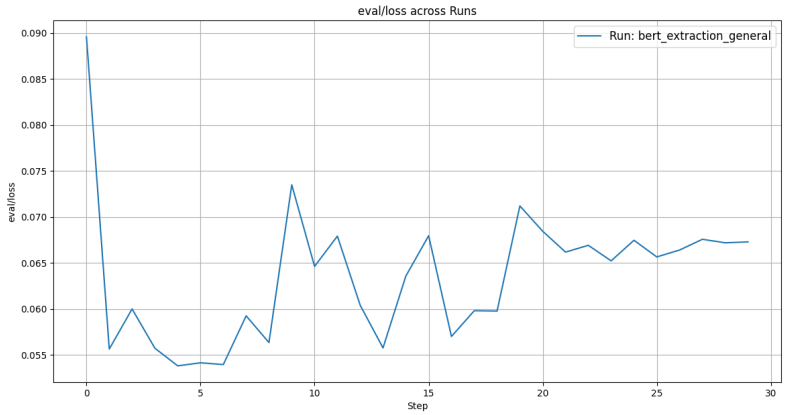

Figure D.1: Eval/loss statistics of extraction pass during general training

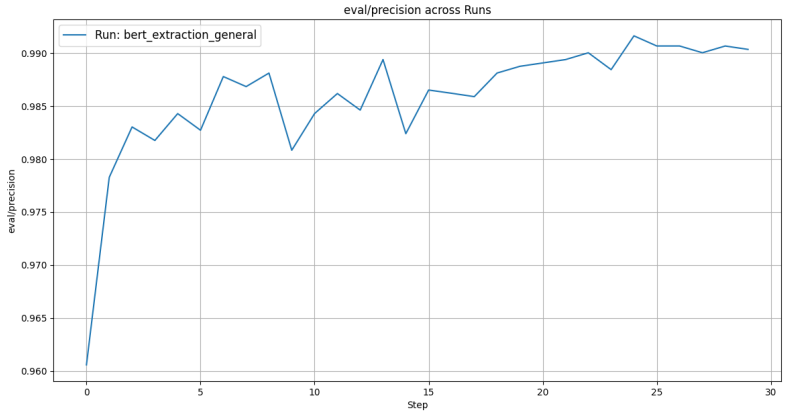

Figure D.2: Eval/precision statistics of extraction pass during general training

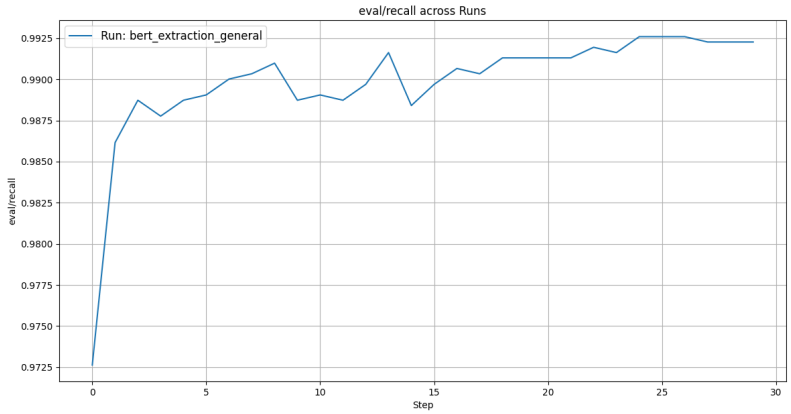

Figure D.3: Eval/recall statistics of extraction pass during general training

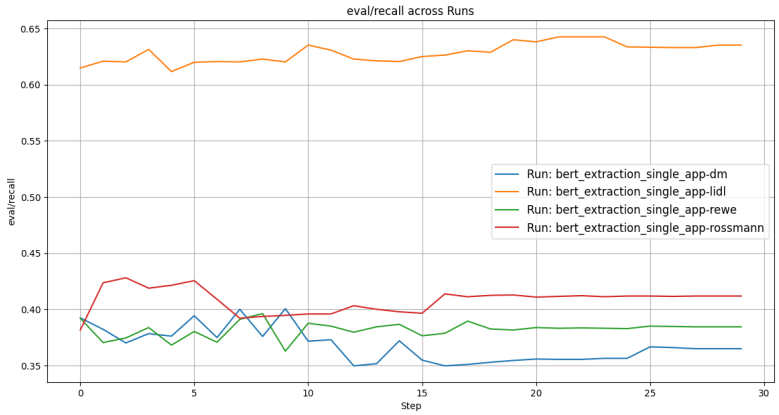

Figure D.4: Eval/recall statistics of extraction pass during separate training

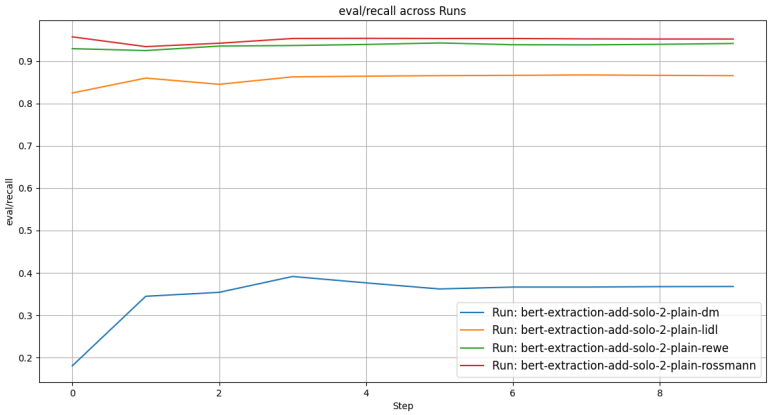

Figure D.5: Eval/recall statistics of extraction pass during incremental separate training

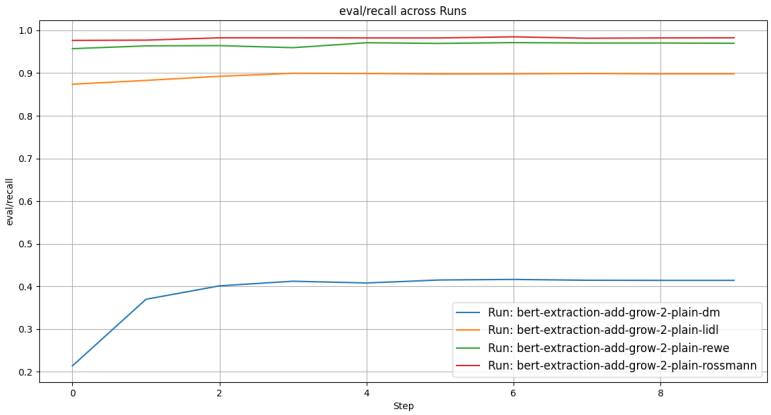

Figure D.6: Eval/recall statistics of extraction pass during incremental combined training

Appendix E

Coupon Similarity Metric

The similarity between coupons is computed by comparing their textual fields using the `ratio()` method of Python's `difflib.SequenceMatcher` class [109]. The similarity computation algorithm is presented in two stages: first, the baseline approach is described, followed by an improved version.

The input to the algorithm consists of an expected coupon and a generated coupon, both containing the text fields `PRODUCT-NAME`, `DISCOUNT-TEXT`, `VALIDITY-TEXT`, and `ACTIVATION-TEXT`. It is assumed that the `PRODUCT-NAME` field is always non-empty.

E.1. Baseline Approach

The similarity of each text field is computed individually. For instance, if `text_1` denotes the value of the `PRODUCT-NAME` field in the expected coupon and `text_2` denotes the corresponding field in the generated coupon, their similarity is computed as follows:

```
similarity = difflib.SequenceMatcher(a=text_1, b=text_2).ratio()
```

Each field is assigned a weight reflecting its relative importance: `PRODUCT-NAME` is weighted 0.4, `DISCOUNT-TEXT` is weighted 0.3, `VALIDITY-TEXT` is weighted 0.2, and `ACTIVATION-TEXT` is weighted 0.1. The final similarity score is calculated as the weighted sum of the individual field similarities.

However, this approach has a notable flaw. Consider the case where the expected coupon has only the `PRODUCT-NAME` field filled with the string `"a"`, while the generated coupon has the `PRODUCT-NAME` field filled with `"b"`; all other fields are empty. Despite the minimal information provided, the resulting similarity score would be 0.6, which overestimates the similarity.

E.2. Improved Algorithm

To mitigate this issue, the improved algorithm modifies how empty fields are handled. Specifically, if a given field is empty in both the expected and generated coupons, its contribution to the final similarity score is ignored. The final similarity score is then rescaled accordingly to reflect only the non-empty fields. For example, if both the `VALIDITY-TEXT` and `ACTIVATION-TEXT` fields are empty in both coupons, their similarities are considered to be 0, and the final similarity score is divided by 0.7, that is, the sum of the remaining fields' weights.

Appendix F

Benchmarking Results

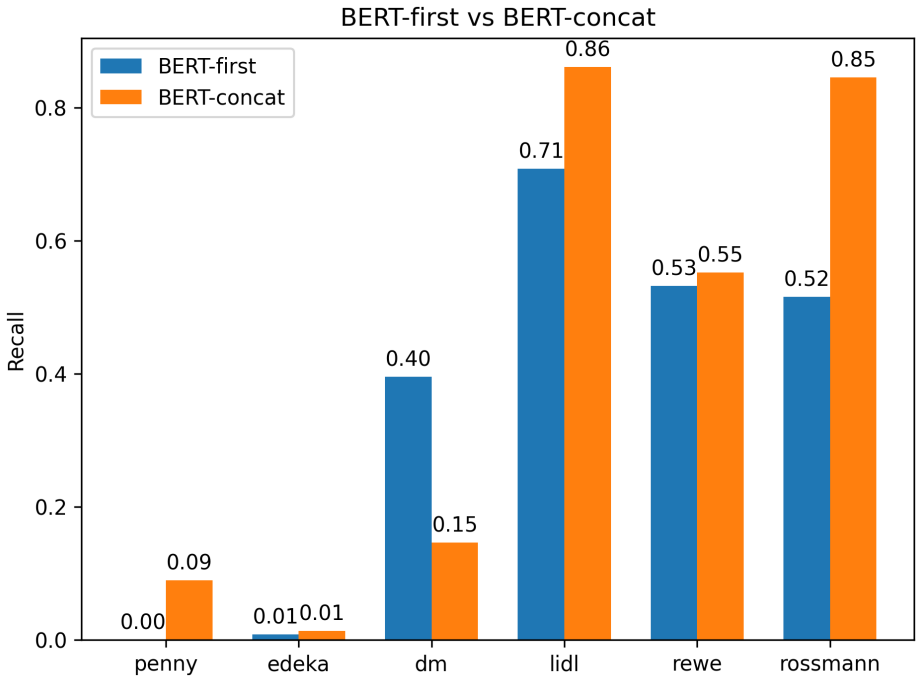

Figure F.1: BERT-first vs BERT-concat - recall

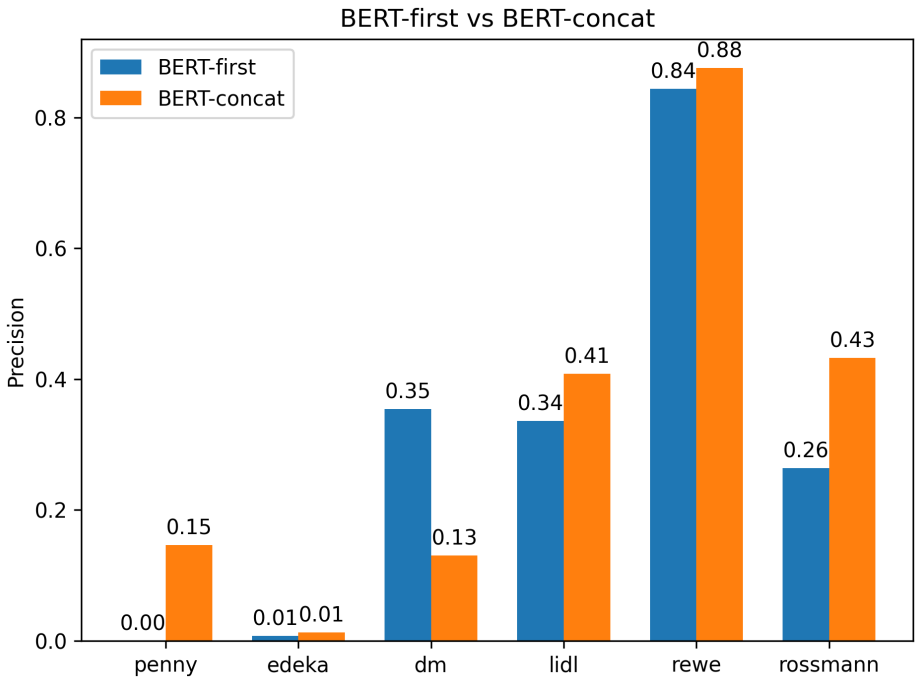

Figure F.2: BERT-first vs BERT-concat - precision

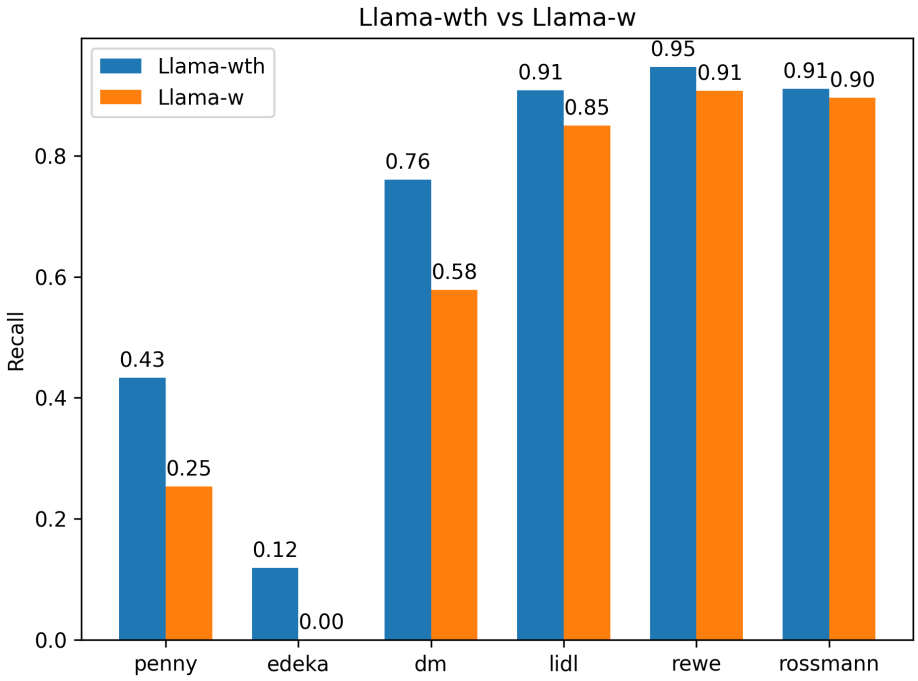

Figure F.3: Llama-wth vs Llama-w - recall

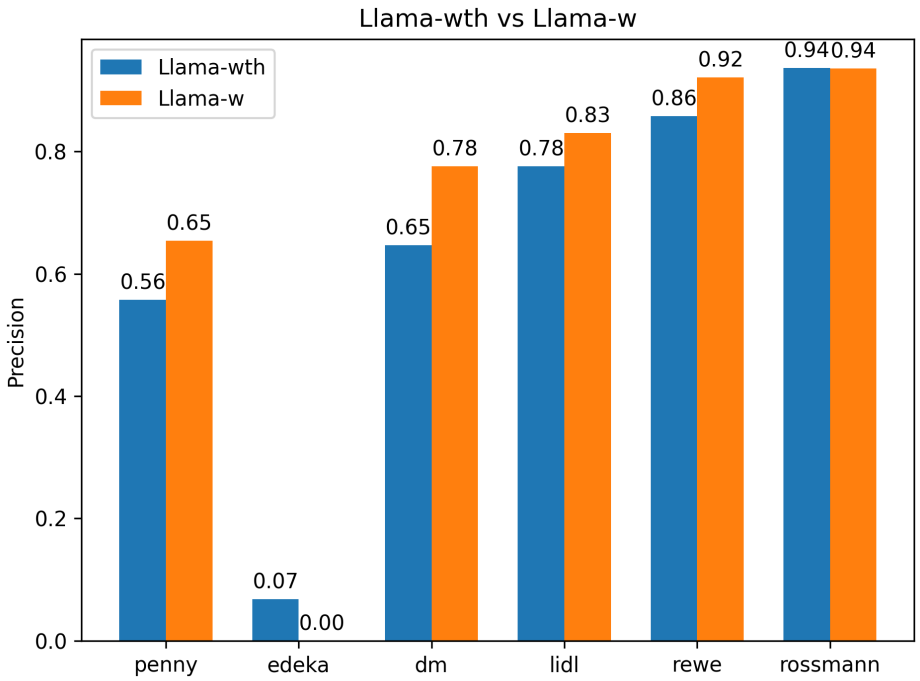

Figure F.4: Llama-wth vs Llama-w - precision

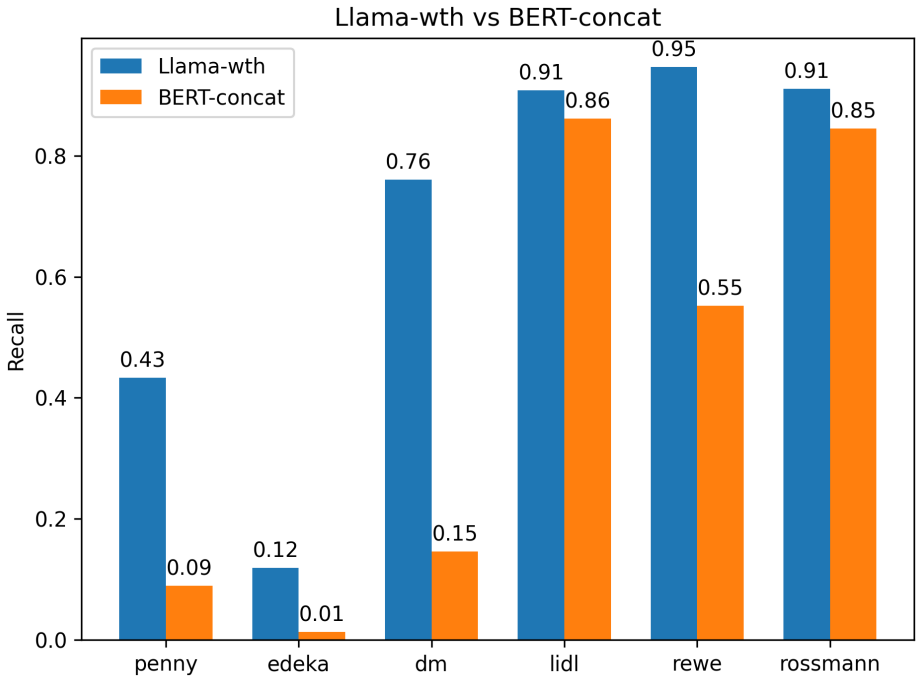

Figure F.5: Llama-wth vs BERT-concat - recall

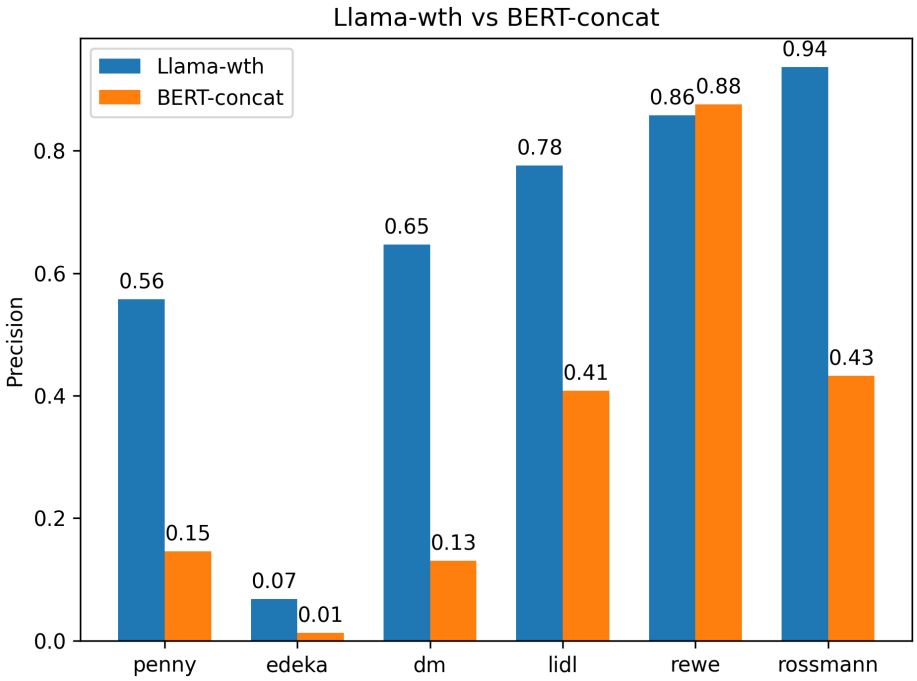

Figure F.6: Llama-wth vs BERT-concat - precision

Appendix G

Comparison of Llama Quantizations

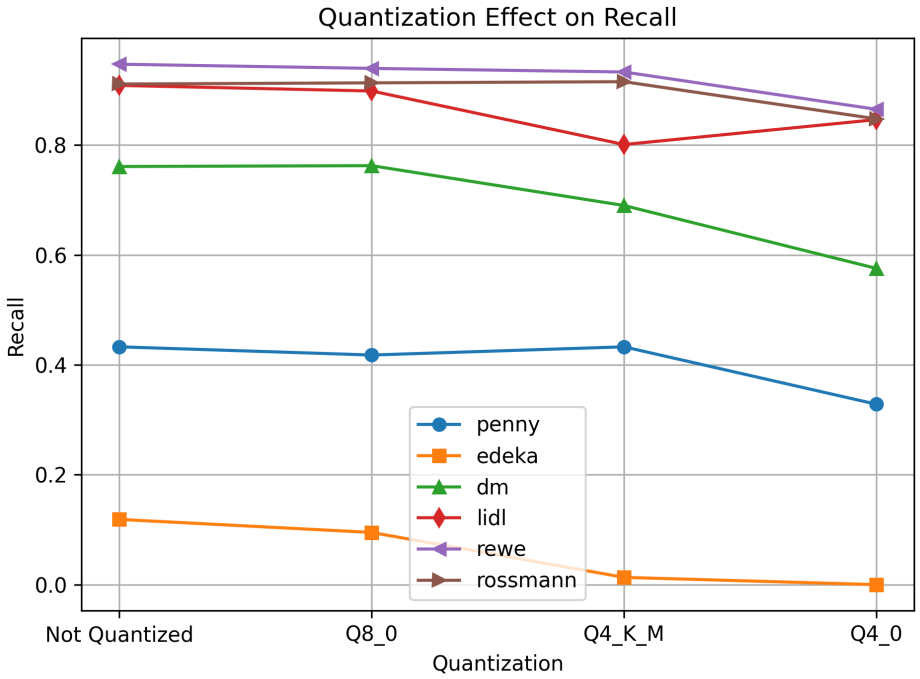

Figure G.1: Llama-wth quantization effect on recall

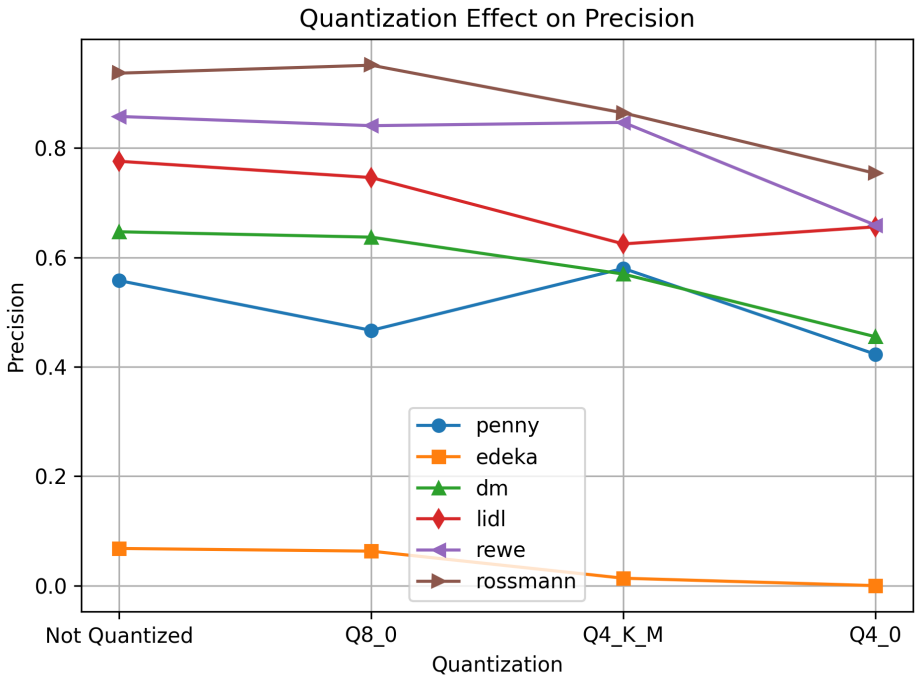

Figure G.2: Llama-wth quantization effect on precision

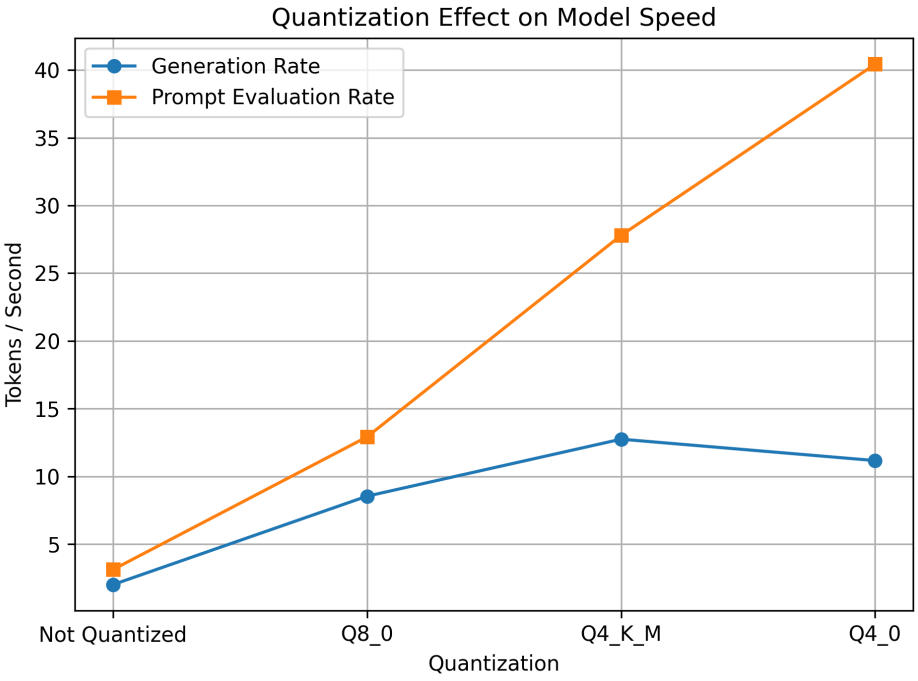

Figure G.3: Llama-wth quantization effect on model speed

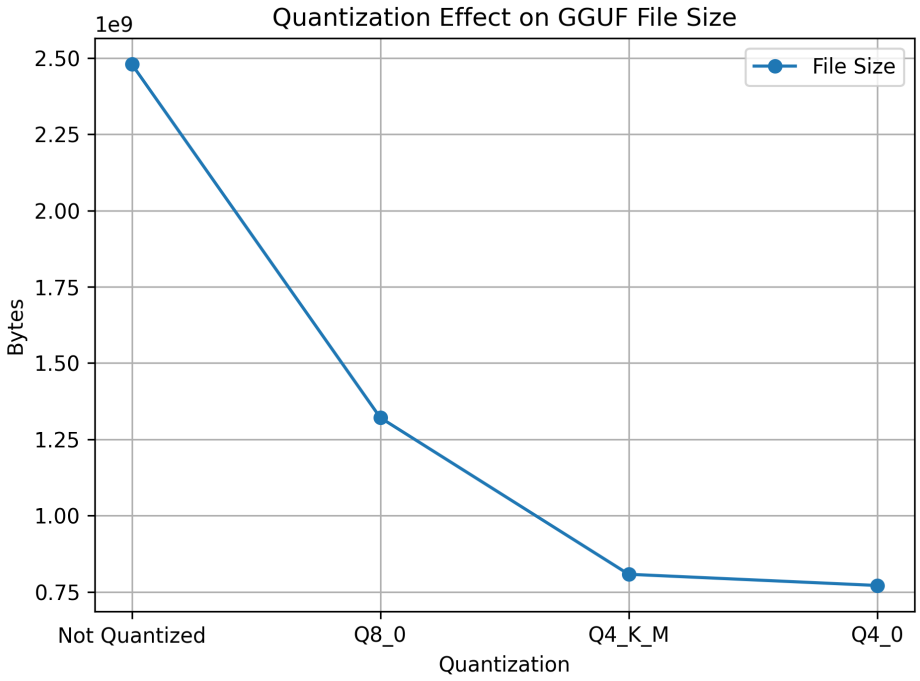

Figure G.4: Llama-wth quantization effect on GGUF file size

Bibliography

[1] *Murmuras website*. https://murmuras.com/. [Accessed 2025-02-11].

[2] *Seo, D., & Yoo, Y. (2023). Improving Shopping Mall Revenue by Real-Time Customized Digital Coupon Issuance. IEEE Access, 11, 7924–7932.* https://doi.org/10.1109/ACCESS.2023.3239425

[3] *Britannica Dictionary definition of COUPON.* https://www.britannica.com/dictionary/coupon. [Accessed 2025-02-03].

[4] *Nayal, P., & Pandey, N. (2021). What Makes a Consumer Redeem Digital Coupons? Behavioral Insights from Grounded Theory Approach. *Journal of Promotion Management*, 28(3), 205–238.* https://doi.org/10.1080/10496491.2021.1989541

[5] *Danaher, P. J., Smith, M. S., Ranasinghe, K., & Danaher, T. S. (2015). *Where, when, and how long: Factors that influence the redemption of mobile phone coupons.* *Journal of Marketing Research*, *52*(5), 710–725.* https://journals.sagepub.com/doi/full/10.1509/jmr.13.0341

[6] *Jayadharshini, P., Sharon Roji, Priya. C, Lalitha, K., Santhiya, S., Keerthika, S., & Abinaya, N. (2023). *Enhancing Retailer Auctions and Analyzing the Impact of Coupon Offers on Customer Engagement and Sales Through Machine Learning.* *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)*, 1–6.* https://doi.org/10.1109/ICCEBS58601.2023.10448900

[7] Xiong Keyi, Yang Wensheng *Research on the Design of E-coupons for Directional Marketing of Two Businesses in Competitive Environment.* https://www.sciencepublishinggroup.com/article/10.11648/j.ijefm.20200801.16. [Accessed 2025-02-04].

[8] *Li, J. (2024). The evolution, applications, and future prospects of large language models: An in-depth overview. Applied and Computational Engineering 35, 234–244.*https://doi.org/10.54254/2755-2721/35/20230399

[9] *Sui, Y., Zhou, M., Zhou, M., Han, S., & Zhang, D. (2024). Table meets LLM: Can large language models understand structured table data? A benchmark and empirical study. Proceedings of the 17th ACM International Conference on Web Search and Data Mining, 645–654.*

[10] Li Li, et. al. *Targeted reminders of electronic coupons: using predictive analytics to facilitate coupon marketing.* https://link.springer.com/article/10.1007/s10660-020-09405-4. [Accessed 2025-02-04].

[11] Bernhard König, et. al. *Analysing competitor tariffs with machine learning.* https://www.milliman.com/en/insight/analysing-competitor-tariffs-with-machine-learning. [Accessed 2025-02-04].

[12] Iqbal H. Sarker *Machine Learning: Algorithms, Real-World Applications and Research Directions.* https://link.springer.com/article/10.1007/s42979-021-00592-x. [Accessed 2025-02-05].

[13] Sara Lebow *How consumers access digital coupons.* https://www.emarketer.com/content/how-consumers-access-digital-coupons. [Accessed 2025-02-05].

[14] *Unveiling IT Coupons Trends and Statistics* https://www.go-globe.com/unveiling-it-coupons-trends-statistics/. [Accessed 2025-02-05].

[15] *Android API Reference - View* https://developer.android.com/reference/android/view/View [Accessed 2025-03-11]

[16] *Brinkmann, A., Shraga, R., Der, R. C., & Bizer, C. (2023). Product Information Extraction using ChatGPT. arXiv preprint, arXiv:2306.14921.* https://arxiv.org/abs/2306.14921

[17] *Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020*

[18] Marco Perini, Lorenzo Padoan, Marco Vinciguerra *Scrapegraph-ai.* https://github.com/VinciGit00/Scrapegraph-ai. [Accessed 2025-02-24].

[19] *Ollama.* https://github.com/ollama/ollama. [Accessed 2025-02-24].

[20] Marco Perini, Lorenzo Padoan, Marco Vinciguerra *Scrapegraph-ai usage.* https://github.com/ScrapeGraphAI/Scrapegraph-ai?tab=readme-ov-file#-usage. [Accessed 2025-02-24].

[21] Marco Perini, Lorenzo Padoan, Marco Vinciguerra *Scrapegraph-ai API and SDKs.* https://github.com/ScrapeGraphAI/Scrapegraph-ai?tab=readme-ov-file#-scrapegraph-api--sdks. [Accessed 2025-02-24].

[22] *Android developer fundamentals website.* https://developer.android.com/guide/components/fundamentals. [Accessed 2025-02-24].

[23] *Apple developer website.* https://developer.apple.com/develop/. [Accessed 2025-02-24].

[24] *Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omni- parser for pure vision based gui agent, 2024.*

[25] *Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., & Wu, Z. (2024). SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. arXiv preprint, arXiv:2401.10935.* https://arxiv.org/abs/2401.10935

[26] Xiang Li, et. al. *Large Language Models on Mobile Devices: Measurements, Analysis, and Insights* https://dl.acm.org/doi/10.1145/3662006.366205

[27] Junchen Zhao, et. al. *LinguaLinked: A Distributed Large Language Model Inference System for Mobile Devices* https://arxiv.org/pdf/2312.00388

[28] *difflib — Helpers for computing deltas* https://docs.python.org/3/library/difflib.html

[29] page 1 *Deep Learning with Python* https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

[30] pages 2 and 3 *Deep Learning with Python* https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

[31] pages 3 and 4 *Deep Learning with Python* https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

[32] pages 7 and 8 *Deep Learning with Python* https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

[33] pages 8 - 10 *Deep Learning with Python* https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf

[34] *What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?* https://blogs.nvidia.com/blog/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

[35] *What is artificial intelligence (AI)?* https://www.ibm.com/think/topics/artificial-intelligence

[36] *Exploring privacy issues in the age of AI* https://www.ibm.com/think/insights/ai-privacy

[37] *MYCIN* https://www.britannica.com/technology/MYCIN

[38] *Supervised versus unsupervised learning: What's the difference?* https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning

[39] *Computer Benchmark.* https://bhatabhishek-ylp.medium.com/benchmarking-in-computer-c6d364681512. [Accessed 2025-02-03].

[40] *The privacy paradox with AI.* https://www.reuters.com/legal/legalindustry/privacy-paradox-with-ai-2023-10-31/. [Accessed 2025-03-24].

[41] *Beware the Privacy Violations in Artificial Intelligence Applications* https://www.isaca.org/resources/news-and-trends/isaca-now-blog/2021/beware-the-privacy-violations-in-artificial-intelligence-applications [Accessed 2025-04-01]

[42] *AI – the threats it poses to reputation, privacy and cyber security, and some practical solutions to combating those threats* https://www.taylorwessing.com/en/global-data-hub/2024/cyber-security---weathering-the-cyber-storms/ai---the-threats-it-poses-to-reputation

[43] *AI has an environmental problem. Here's what the world can do about that.* https://www.unep.org/news-and-stories/story/ai-has-environmental-problem-heres-what-world-can-do-about [Accessed 2025-03-24]

[44] *Top Use Cases of AI-Based Recommendation Systems.* https://www.itconvergence.com/blog/top-use-cases-of-ai-based-recommendation-systems/. [Accessed 2025-03-24].

[45] *14 Risks and Dangers of Artificial Intelligence (AI).* https://builtin.com/artificial-intelligence/risks-of-artificial-intelligence. [Accessed 2025-03-24].

[46] *The growing data privacy concerns with AI: What you need to know.* https://www.dataguard.com/blog/growing-data-privacy-concerns-ai/. [Accessed 2025-03-24].

[47] *Examining Privacy Risks in AI Systems.* https://transcend.io/blog/ai-and-privacy. [Accessed 2025-03-24].

[48] *Exploring privacy issues in the age of AI.* https://www.ibm.com/think/insights/ai-privacy. [Accessed 2025-03-24].

[49] *Deep Learning's Carbon Emissions Problem.* https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/. [Accessed 2025-03-24].

[50] *The growing energy footprint of artificial intelligence.* https://www.researchgate.net/publication/374598219_The_growing_energy_footprint_of_artificial_intelligence. [Accessed 2025-04-16].

[51] *A.I. Could Soon Need as Much Electricity as an Entire Country.* https://www.nytimes.com/2023/10/10/climate/ai-could-soon-need-as-much-electricity-as-an-entire-country.html. [Accessed 2025-03-24].

[52] *Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning.* https://www.sciencedirect.com/science/article/pii/S2210537923000124#sec7. [Accessed 2025-03-24].

[53] *A Computer Scientist Breaks Down Generative AI's Hefty Carbon Footprint.* https://www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/. [Accessed 2025-03-24].

[54] *AI Is Accelerating the Loss of Our Scarcest Natural Resource: Water.* https://www.forbes.com/sites/cindygordon/2024/02/25/

ai-is-accelerating-the-loss-of-our-scarcest-natural-resource-water/. [Accessed 2025-03-24].

[55] *AI has an environmental problem. Here's what the world can do about that..* https://www.unep.org/news-and-stories/story/ai-has-environmental-problem-heres-what-world-can-do-about. [Accessed 2025-03-24].

[56] *What is deep learning?*. https://www.ibm.com/think/topics/deep-learning. [Accessed 2025-03-24].

[57] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* https://arxiv.org/abs/1810.04805v2. [Accessed 2025-04-04]

[58] *BERT model page on hf.* https://huggingface.co/google-bert/bert-base-uncased. [Accessed 2025-04-04]

[59] *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* https://arxiv.org/abs/1506.01497 [Accessed 2025-04-04]

[60] *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* https://arxiv.org/abs/1907.11692 [Accessed 2025-04-04]

[61] *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.* https://arxiv.org/abs/1909.11942 [Accessed 2025-04-04]

[62] *ALBERT model page on hf.* https://huggingface.co/albert/albert-base-v2. [Accessed 2025-04-04]

[63] *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter* https://arxiv.org/abs/1910.01108 [Accessed 2025-04-04]

[64] *Comparative Analysis of BERT Variants for Text Detection Tasks* https://www.researchgate.net/publication/385142549_Comparative_Analysis_of_BERT_Variants_for_Text_Detection_Tasks [Accessed 2025-04-04]

[65] *Facebook AI on HuggingFace* https://huggingface.co/FacebookAI [Accessed 2025-04-04]

[66] *Multilingual BERT on HuggingFace* https://huggingface.co/google-bert/bert-base-multilingual-cased [Accessed 2025-04-04]

[67] *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference* https://arxiv.org/abs/2412.13663 [Accessed 2025-04-05]

[68] *ModernBERT on HiggingFace* https://huggingface.co/answerdotai/ModernBERT-base [Accessed 2025-04-05]

[69] *Curriculum learning* https://dl.acm.org/doi/abs/10.1145/1553374.1553380 [Accessed 2025-04-07]

[70] The echo chamber effect on social media https://pubmed.ncbi.nlm.nih.gov/33622786/ [Accessed 2025-04-09]

[71] Energy and Carbon Considerations of Fine-Tuning BERT https://arxiv.org/html/2311.10267 [Accessed 2025-04-09]

[72] What is fine-tuning? https://www.ibm.com/think/topics/fine-tuning [Accessed 2025-04-09]

[73] 14.2. Fine-Tuning https://d2l.ai/chapter_computer-vision/fine-tuning.html [Accessed 2025-04-09]

[74] What is quantization? https://www.ibm.com/think/topics/quantization/ [Accessed 2025-04-09]

[75] Quantization https://huggingface.co/docs/optimum/en/concept_guides/quantization [Accessed 2025-04-09]

[76] Float32 vs Float16 vs BFloat16? https://newsletter.theaiedge.io/p/float32-vs-float16-vs-bfloat16 [Accessed 2025-04-09]

[77] Attention Is All You Need https://arxiv.org/pdf/1706.03762 [Accessed 2025-04-08]

[78] Exploring Multi-Head Attention: Why More Heads Are Better Than One https://medium.com/%40hassaanidrees7/exploring-multi-head-attention-why-more-heads-are-better-than-one-006a5823372b [Accessed 2025-04-08]

[79] Understanding the Transformer Model: A Report on "Attention Is All You Need" by Ashish Vaswani et al. https://medium.com/%40shivayapandey359/attention-is-all-you-need-26586e6ab8ca [Accessed 2025-04-08]

[80] *Pareto principle* https://en.wikipedia.org/wiki/Pareto_principle [Accessed 2025-04-10]

[81] *Named Entity Recognition* https://cs229.stanford.edu/proj2005/KrishnanGanapathy-NamedEntityRecognition.pdf [Accessed 2025-04-10]

[82] *Llama 3.2 published by Meta* https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/ [Accessed 2025-04-17]

[83] *Unsloth* https://unsloth.ai/ [Accessed 2025-04-17]

[84] *The Hugging Face Platform* https://huggingface.co/ [accessed 17.04.2025]

[85] *The Github platform* https://github.com/ [accessed 17.04.2025]

[86] *The Modal platform* https://modal.com/ [accessed 17.04.2025]

[87] *The Wandb platform* https://wandb.ai/ [accessed 17.04.2025]

[88] *The website of Python programming language* https://www.python.org/ [accessed 17.04.2025]

[89] *Our experiment with mobile app that runs in background* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/tree/main/research/service_demo_app

[90] *Webpage of the Kotlin programming Language* https://kotlinlang.org/ [accessed 24.04.2025]

[91] *dward J. Hu and Yelong Shen and Phillip Wallis and Zeyuan Allen-Zhu and Yuanzhi Li and Shean Wang and Lu Wang and Weizhu Chen (2021): LoRA: Low-Rank Adaptation of Large Language Models* https://arxiv.org/abs/2106.09685

[92] *Tillet, Philippe and Kung, H. T. and Cox, David (2019): Triton: an intermediate language and compiler for tiled neural network computations* https://doi.org/10.1145/3315508.3329973

[93] *Lhoest et al (2021): Datasets: A Community Library for Natural Language Processing* https://arxiv.org/abs/2109.02846

[94] *evaluate Python library* https://pypi.org/project/evaluate/ [accessed 24.05.2025]

[95] *Wolf et al (2020): Transformers: State-of-the-Art Natural Language Processing* https://aclanthology.org/2020.emnlp-demos.6/

[96] *Llama.cpp repository* https://github.com/ggml-org/llama.cpp [Accessed 25.04.2025]

[97] *SpaCy Python library* https://pypi.org/project/spacy/ [Accessed 25.04.2025]

[98] *ONNX framework* https://github.com/onnx/onnx [Accessed 25.04.2025]

[99] *LiteRT environment* https://ai.google.dev/edge/litert [Accessed 25.04.2025]

[100] *ExecuTorch framework* https://pytorch.org/executorch-overview [Accessed 25.04.2025]

[101] *Android Studio webpage* https://developer.android.com/studio [Accessed 25.04.2025]

[102] *Jupyter Notebook Webpage* https://jupyter-notebook.readthedocs.io/en/stable/ [Accessed 25.04.2025]

[103] *IPython Kernel Webpage* https://ipython.org/ [Accessed 25.04.2025]

[104] *Overview od spacy* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/tree/main/research/spacy_research [Accessed 25.04.2025]

[105] *Experiments with Scrapegraph AI* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/tree/main/research/scrapegraphai [Accessed 25.04.2025]

[106] *Demonstrative mobile deployment of a model in ONNX/ framework.* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/tree/main/research/onnx_demo_app [Accessed 25.04.2025]

[107] *Demostrative mobile deployment of LiteRT model.* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/tree/main/research/litert_deployment [Accessed 25.04.2025]

[108] *Overview of ExecuTorch suitability in our problem.* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/blob/main/research/executorch/executorch_llama_report.md [Accessed 25.04.2025]

[109] *difflib — Helpers for computing deltas* https://docs.python.org/3/library/difflib.html [Accessed 28.04.2025]

[110] *Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, Jian Guo (2025): A Survey on LLM-as-a-Judge* https://doi.org/10.48550/arXiv.2411.15594

[111] *Georgi Gerganov and contributors: Quantization Methods in llama.cpp* https://github.com/ggml-org/llama.cpp/blob/master/examples/quantize/README.md