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

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

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
Jacek Sroka PhD
Institute of Informatics

Warsaw, May 25, 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 two novel approaches to processing phone screen content, such as displayed social media posts and website advertisements. Our solutions leverage 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 methods in identifying and structuring valuable content from varying digital sources. Furthermore, these systems are designed to be easily adaptable to other use cases, such as analyzing users' political views. Additionally we explore 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, LLama, 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

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 [120], replacing previously utilized recurrent neural network (RNN) architectures in the field of natural language processing (NLP) [8]. Subsequent research [9] has demonstrated their applicability to structured input data, 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. In response, we proposed two solutions: one utilizing two BERT [57] models, and the second one using one Llama model. Both approaches will process input representing the user's onscreen view and extract digital coupons along with their relevant data. Thess solutions 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 1.1

(a) Example coupon from fast-food restaurant app

(b) Example coupon from grocery store app

Figure 1.1: Example digital coupons

1.3. The Significance of the digital coupon

The digital coupon is one of the most important tools in contemporary marketing strategies [10], therefore analyzing their life cycle 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 competitor's digital coupon strategies enables businesses to identify market trends, adjust their promotional tactics, and maintain a competitive edge in the evolving digital marketplace.

1.4. Murmuras approach

The measurement of statistics such as 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. By paying users to install tools for tracking their screen activity, and to share this data, Murmuras is able to directly collect coupon-related data from users' devices. Having access to the real-time access to the contents of the user's screen, Murmuras can gather data using screen content scraping tool, that can be later used to calculate coupon redemption rates but also the percentage of users that were presented with specific coupons and many more. This allows for large-scale acquisition of real-world data.

1.5. 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 following format:

1. PRODUCT-NAME: the name of the product,
2. ACTIVATION-TEXT: some coupons can contain text that informs us whether the coupon was activated or not. This can be used to calculate the popularity of a specific coupon.
3. DISCOUNT-TEXT: the text representing the discount offered to the user,
4. VALIDITY-TEXT: text specifying the expiration date of the coupon.

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",
    "ACTIVATION-TEXT": "Coupon activated",
    "DISCOUNT-TEXT": "20% OFF",
    "VALIDITY-TEXT": "Valid until 2025-06-30"
}
```

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.

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.

1.6. 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. This tool should be capable of handling various coupon formats and layouts with high accuracy;
3. A pipeline for preparing the input data for the previously mentioned tool, and for post-processing its output into a common format;
4. A key requirement is that the machine learning model must be deployable on the mobile device itself. By performing the inference locally, and sending only results of this inference to the server, we guarantee the data privacy.

1.7. Potential applications of the project

1.7.1. 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 be utilized in monitoring of competitors' coupon strategies, their effectiveness, and whether they provide better discounts. As an example, many online shops function as resellers of different brands. By analyzing coupons of these brands, they can decide which products needs additional marketing and/or 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.

1.7.2. Acquiring of data for research purposes

In the age of smartphones and ubiquitous internet, more and more resources are being invested in research regarding social impacts of those technologies. One of many aspects of our social life is shopping. With a tool that can collect data about the presence and popularity of different coupons without depending on the publisher of those, it will be much easier and faster for researchers to gather real-world data regarding e.g. customer habits or the percentage of coupons that an average user clicks.

1.8. Work structure

In the next chapters, we first describe dangers related to the use of AI, focusing on privacy and environmental concerns. Then, we take a look at approaches similar to ours, that not necessarily utilize LLMs. After that, we describe datasets we used in our work, as well as technologies used in our experiments. Finally, we describe both our approaches, and summarize the whole project.

Chapter 2

Machine learning and the dangers associated with it

Artificial intelligence (AI) has seen intense media coverage in recent years, with discussions ranging from transformative applications (e.g., autonomous vehicles, virtual assistants) to growing societal concerns. Key issues include workforce automation [29], privacy risks from training on sensitive personal data [36], and potential compromises to secure communication protocols like End-to-End Encryption [121].

This chapter first distinguishes between Artificial Intelligence, Machine Learning, and Deep Learning. We then detail the transformer architecture used in our implementation, followed by analysis of privacy implications and environmental impacts of AI systems.

2.1. AI, machine learning, and deep learning

AI encompasses systems performing human-like tasks, including machine learning (ML) and deep learning (DL) [35, 30]. ML enables systems to learn without explicit rules, while DL uses neural networks for this purpose [35]. Their hierarchical relationship is shown in Figure 2.1.

2.1.1. Artificial intelligence

AI automates human intellectual tasks using various approaches. While machine learning and deep learning rely on data-driven models, symbolic AI uses predefined rules—like the MYCIN

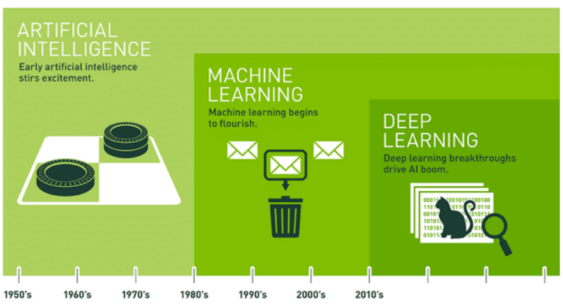

Figure 2.1: The hierarchy of artificial intelligence, machine learning and deep learning [34]

expert system, which diagnosed infections using IF-THEN rules with specialist-level accuracy [30, 37].

2.1.2. Machine learning

Machine learning (ML) trains systems using data rather than explicit programming [31]. The two main approaches are:

- **Supervised learning**: Models learn from labeled data, mapping inputs to correct outputs. For instance, coupon extraction models (Fig. 2.2) are trained on text paired with labels (product, price, discount, etc.).
- **Unsupervised learning**: Models identify patterns in unlabeled data.

```
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.
```

(a) textual representation

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

(b) structured representation

Figure 2.2: Example of textual (a) and structured (b) representations of the coupon.

2.1.3. Supervised vs. unsupervised learning

Supervised learning excels at classification (e.g., spam detection) and regression (e.g., sales forecasting), using labeled data for precise predictions.

Unsupervised learning identifies patterns in unlabeled data, such as product purchase clusters [38]. Techniques like Principal Component Analysis (PCA) [122] reduce dimensionality by projecting data onto orthogonal axes of maximum variance.

For this project, supervised learning is optimal as our coupon extraction task—identifying and classifying elements in XML trees—requires labeled training data rather than pattern discovery. This approach ensures accurate recognition and data extraction.

2.1.4. 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 2.3; each representation is increasingly different from the original and more useful to determining the result. Traditionally, while machine learning models focus

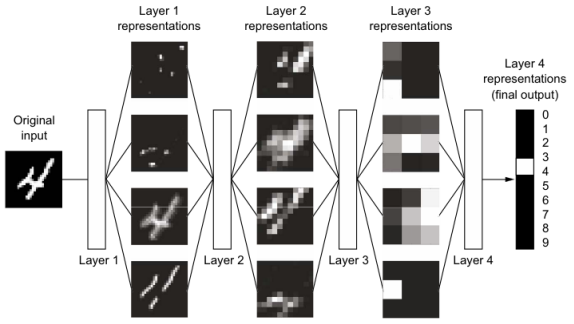

Figure 2.3: Data representations learned by a digit-classification model [32]

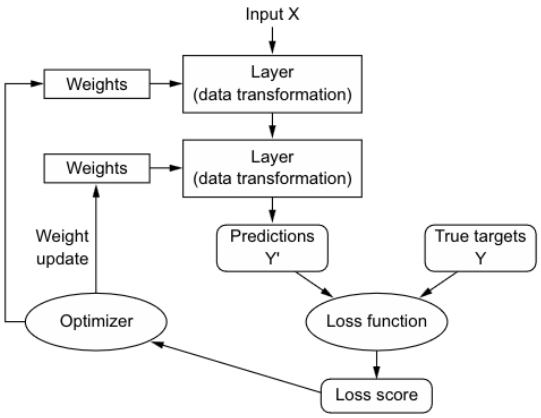

Figure 2.4: A visualization of how a Deep Learning model works [33]

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 2.4 [33].

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

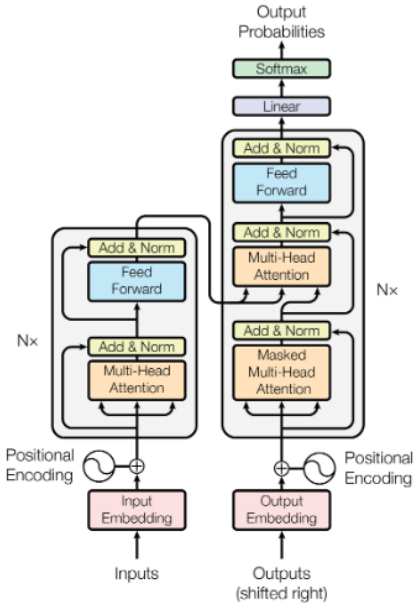

Figure 2.5: Transformer Architecture [77]

2.1.5. 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 [123], transformers utilize self-attention mechanisms to process input data in parallel, enhancing efficiency and scalability. This architecture has become foundational in models like BERT, and GPT [79]. An example overview of this architecture can be seen on 2.5. For more technical details, please refer to the "Chapter 3" of the aforementioned paper.

2.1.6. 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 $\rightarrow$ float16 and float32 $\rightarrow$ int8 [75].

float32 $\rightarrow$ 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, i.e. whether it is

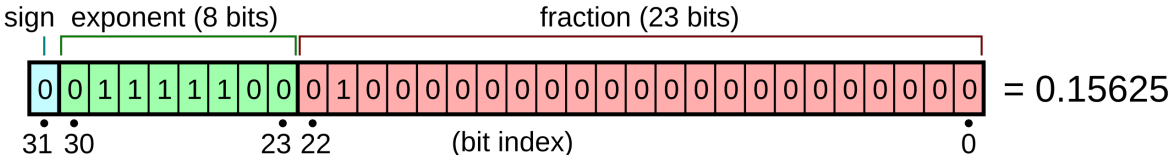

Figure 2.6: Example representation of float32 [124]

negative or positive.

2. The next 8 bits represent the exponent.

3. The remaining 23 bits, are the mantissa, i.e. the decimal.

According to the IEEE Standard for Floating-Point Arithmetic (IEEE 754) [124], for 32 bits this format can represent numbers of which the absolute value lies between $1.18 * 10^{-38}$ and $3.40 * 10^{38}$. An example representation can be seen here on figure 2.6. When converting from float32 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 $\rightarrow$ 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 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].

2.1.7. fine-tuning

fine-tuning adapts pre-trained models to specific tasks (e.g., coupon extraction) using task-specific datasets. This transfer learning approach is particularly valuable for deep learning models (LLMs, CNNs) as it:

- Prevents overfitting common when training large models on small datasets
- Reduces computational resources and data requirements [72]

The fine-tuning process involves:

1. Starting with a model pre-trained on a large general dataset

2. Replacing the final output layer with task-specific architecture

3. Training on target data while adjusting pre-trained weights [73]

For our coupon extraction task, fine-tuning allows leveraging general visual/textual features while specializing for coupon-specific elements.

2.2. AI Risks and ethical considerations

The rapid advancement of AI raises legitimate concerns about:

- Privacy and surveillance risks from pervasive monitoring systems [40, 41, 42]
- Environmental impact due to high energy/water consumption and rare material requirements [43]

This section evaluates these risks and discusses mitigation strategies implemented in our solution.

2.2.1. Privacy erosion in AI systems

Modern AI's hunger for data creates a fundamental tension between personalization and privacy. At the heart of this lies machine learning's need for vast training datasets - often containing sensitive health records and biometric information [48, 46]. These systems don't just process what we explicitly share; they uncover what we don't. Through predictive harm, AI can deduce intimate details like political leanings or sexual orientation from seemingly innocuous data points [47], creating profiles far more revealing than users anticipate.

The Cambridge Analytica scandal demonstrated this danger vividly. What began as personality quizzes for 87 million Facebook users became a blueprint for mass behavioral manipulation [47]. This case revealed how AI can weaponize ordinary digital interactions, turning "nothing to hide" into "everything to lose" when systems infer more than we intend to share.

Our approach counters these risks through architectural design. By processing data exclusively on users' devices with locally-deployed models, we create an AI solution that never requires sensitive data to leave its source. This on-device paradigm prevents the surveillance risks of cloud-based systems while maintaining full functionality - proving effective AI needn't come at privacy's expense.

2.2.2. Environmental impact of AI

The rapid expansion of artificial intelligence systems has revealed significant ecological costs, particularly in terms of energy and water consumption. Current projections indicate that by 2027, AI-related electricity demand could reach 85–134 TWh annually [50], representing approximately 0.5% of global consumption. This estimate stems from the dominant use of Nvidia A100 servers in AI infrastructure [51].

The carbon footprint of training large language models is especially concerning. For instance:

- Training BERT consumed energy equivalent to a transatlantic flight
- GPT-3's training emitted 552 metric tons of CO_2, comparable to annual emissions from 123 gasoline-powered vehicles [53]

Water usage presents another critical challenge. AI data centers consume up to 9 liters of water per kWh for cooling systems [55]. With projections suggesting AI infrastructure may soon require six times Denmark's annual water consumption, these demands conflict directly with global needs - particularly as the UN estimates half the world's population will face water stress by 2030 [54].

Sustainable Approach

Our solution addresses these environmental concerns by fine-tuning existing models rather than training them from zero. According to the findings of Wang et al., BERT fine-tuning requires 400–45,000× less energy than the full training. This way, we greatly limit our CO_2 emission as well as water consumption.

Chapter 3

Related problems

To the best of our knowledge, at the time of commissioning of this project, no publicly available solutions directly addressed this problem. The most comparable approaches involve existing multi-modal 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 programs 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. Web scraping

With XML tree structure being similar to the HTML tree, our problem can be seen as one from web scraping domain. Web scraping [125] is a process of extracting data from a website. This usually involves downloading its HTML code, and parsing its structure.

3.2.1. Code-based scraping

A common approach to this problem would be writing code that would first download the HTML page and then parse it. Over the years, many libraries were created for this task, such as BeautifulSoup4 [126]. It was written on top of HTML and XML parser, making it easy for user to reference specific fields by their ids, classes or HTML tags. From our perspective however, this solution has one mayor drawback: it's very difficult to scale it for multiple sources. Usually, scrapers of this type are being made for a specific site. With each shopping application/website having a different coupon format, it would be impossible to extend our solution for more that few data sources.

3.2.2. AI-based scraping

Since the public release of ChatGPT-3.5 in 2022, chat-based models came a long way. From the scraping perspective, one of the most useful addition is an option to attach files into the conversation, instead of having to copy-paste them into the chat. This functionality is supported by e.g. ChatGPT and DeepSeek [127]. However, from the perspective of our problem, this solution has a drawback of not being automatable.

ChatGPT seems to counter this issue. With OpenAI API [128] available, programmers can easily write Python code that will request LLM responses to their queries. It also provides much more customizability of the query than the chat, having e.g. an option to specify the model role, response length etc. This service was even used in the Murmuras PoC solution. The main problem with this approach is its price. Requests performed through the API require OpenAI credits, which have to be purchased with real money. So while its a cheap option for testing ideas, production usage would prove to be too costly for a company such as Murmuras.

To reiterate, it is important to notice, that even without the described issues, this approach would not be suitable for us, because those mentioned models are running on outside servers. Sending them data to analyze would compromise data privacy, which is one of our main concerns.

3.2.3. Scapegraph AI

ScrapeGraphAI[18] is an open-source library that streamlines data extraction from websites and local documents by utilizing LLMs and graph logic to construct scraping pipelines. 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 4.2), and the *coupons* CSV file, which serves as the ground truth (see 4.1). 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 removed unnecessary columns from the datasets.

content_full
[Treuepunkte, 10% Rabatt*, auf alle REWE Bio + vegan Produkte!, Kaufe mind. 2 Produkte., Gültig bis 14.01.2024, Coupon aktivieren]

discount_text	**activation_text**	**validity_text**
10% Rabatt*	Coupon aktivieren	Gültig bis 14.01.2024

product_text	**discount_details**	**aggregation**
auf alle REWE Bio + vegan Produkte!	Kaufe mind. 2 Produkte.	1704181455161

Table 4.1: Cleaned coupon file format (ground truth)

view_id	**text**	**view_depth**	**aggregation**
de.rewe.app.mobile:id/action_bar_root		3	1704181453677

Table 4.2: Screen content file format

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. The results of this preprocessing are described in the following sections.

4.2.1. BERT datasets

The BERT models accept two dataset formats during the coupon selection phase in which the model identifies where a coupon begins and ends:

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

On the other hand, the format depicted in Table 4.3 is used during the coupon extraction phase, wherein relevant coupon information is tagged with structured labels. Each label corresponds to a specific type of information extracted from the screen text:

- **0** – Generic or descriptive text
- **1** – Product or item the coupon is about
- **3** – Coupon activation phrase
- **5** – Validity date
- **7** – Discount or offer

Text	**Label**
tiefgefroren je 410–460 g Packung (1kg= 6.50/7.29)	0
Coupon aktivieren	3
Gültig bis 07.01.2024	5
Knaller 2.99 € statt Aktionspreis 3.33 €!*	7
Gustavo Gusto Pizza Margherita oder Salame	1

Table 4.3: Example of text and corresponding labels in the coupon information extraction dataset.

```
{
   "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 task description to the Llama model.
2. **one_input_multiple_outputs_wthrequest**: the dataset does not include a task description.

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

```
{
   You are provided with text representing contents of the
   phone screen. Your task is to extract information about
   coupons from the text. The information should include
   the product name, the validity text, the discount text
   and the activation text.
   ### Input: {screen content data}
   ### Response: {coupons in a form of JSON array}
}
```

Figure 4.2: Llama input format with a task description

```
{
   ## Input: {screen content data}
   ## Response: {coupons in a form of JSON array}
}
```

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.

As seen in 4.7, 92.68% of Edeka coupons and 50% of DM and Penny coupons, are missing at least one feature. This missing data in coupon features presents clear challenges for building reliable extraction models as it makes it difficult for models to learn consistent patterns across sources.

A major consequence is lower recall. When key fields such as discount text, product names, or validity dates are missing during training, the model lacks enough examples to learn how to identify them reliably. This often results in false negatives, especially for sources with frequent data gaps, while performance tends to improve on more complete datasets.

This presents evaluation challenges, as we have to account for the possibility that some ground truth coupons are unannotated. As a result, a model may correctly identify a valid instance but be penalized as if it were a false positive or false negative, which can distort metrics like precision, recall, and F1-score.

Application	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 4.4: Dataset sizes and example counts per application and split for the coupon selection stage.

Application	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 4.5: Dataset size and example counts per application and split for the Llama models

Application	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 4.6: Dataset size and example counts per application and split for the BERT models

Application	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 4.7: Missing feature counts and percentage of coupons missing at least one feature per company (test and train sets combined)

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 accessing models, as well as storing and managing both their trained versions and datasets used in training. 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, as well as mobile deployment tools including ONNX[98, 106], liteRT[99, 107], and Executorch[100, 108]. These experiments were conducted 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.

In our approach, inspired by region proposal networks in computer vision [59], two BERT models were used. The first one identifies coupon regions, while the second extracts their attributes.

6.1. 2-Stage architecture

Figure 6.1 illustrates the system's pipeline. The process begins with input data captured as XML trees from phone frames (See 1), which are flattened into timestamp-grouped CSV entries with concatenated text.

The selection stage employs a multilingual BERT model for coarse NER classification, tagging tokens as either `COUPON` or `UNKNOWN` using IOB2 format [81]. The model outputs are processed into contiguous strings, with only `COUPON`-labeled segments progressing to the second stage.

In the extraction phase, another multilingual BERT model performs fine-grained NER, classifying coupon tokens into four specific fields according to our data model (1.1). The final output assembles these tokens into strings by field labels and packages them as JSON objects.

In some cases, the extraction model identifies multiple entities with the same label. To address this, we evaluated several strategies for resolving such ambiguities. Our primary approach takes advantage of the fact that the BERT model provides confidence scores (probabilities) along with the predicted NER tags. Specifically, we select the entity with the highest confidence score, breaking ties by favoring the entity that appears earlier in the text. We refer to this variant of the pipeline as `BERT-top_score`.

Additionally, we experimented with two alternative strategies: one that concatenates all entities with the matching label (`BERT-concat`), and another that selects the first entity with the given label in the text (`BERT-first`).

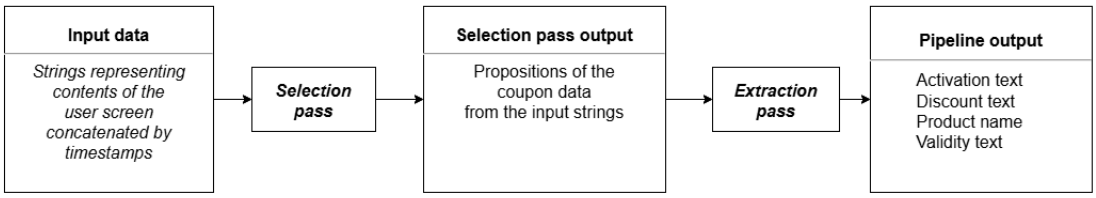

Figure 6.1: Pipeline graph

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

Listing 1: Pipeline input

6.2. 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 of 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 with 179M parameters [66]. This decision was driven mostly by the fact that all of the other mentioned models lacked multilingual support, making BERT-base really the only model that would only require fine-tunings on our data to work.

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. In total, we conducted four different experiments:

1. **General Fine-tuning**:

   Main goal of this experiment was to establish baseline performance metrics. Additionally, this experiment was used as a basis to abandon Curriculum learning and JSON format approaches. It was conducted by fine-tuning models on combined data from all four applications.

2. **Per-application Fine-tuning**:

   In this experiment, we wanted to examine cross-application generalization capabilities of our pipeline. To do this, we fine-tuned individual models separately on each application's data, and then tested them on other applications to measure similarities between datasets. This measurement was based on how well model trained on app X behaves on test dataset containing data from all four apps.

3. **Incremental Separate Fine-tuning**:

   This was the first of two experiments used to see, how easily our solution could be generalized/extended. To conduct it, we performed four consecutive fine-tunings on each model:

   (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

   This way, we were able to see whether models gain more knowledge when presented with new sources of data or they loose it.

4. **Incremental Combined Fine-tuning**:

   Idea behind our last experiment was similar to the Incremental Separate Fine-tuning, but this time we were adding new data to the training dataset, instead of changing them. This was intended as a way to prevent models from forgetting previously learned patterns. For that, we performed the following fine-tunings:

(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** [130]:

   When working with data grouped by timestamps, we loose the information about the relations between different elements on the user's screen. With XML tree preservation, we wanted to see if maintaining this information would improve coupon detection. To perform this experiment, we created special datasets, where screen content strings were wrapped in JSON structures simulating the XML hierarchy.

   Ultimately, we decided to abandon this approach for two reasons: first of all, it was performing worse than the original approach, called *plain-text* (see plots A.1, B.2, A.3). Second of all, wrapping everything in JSON structure introduced more text, which lead to bigger dataset sizes. As a result, fine-tunigs were longer and more costly, with no clear benefit.

2. **Token Distribution Balancing** [129]:

   First batches of data that we received from Murmuras had skewed data distribution. For many timestamps, only 10 % of words contained in them were part of any coupon. As a result, models were not learning much, because labeling everything as UNKNOWN (not belonging to any coupon) was enough for more than 90 % precision and recall. To counter this issue, we implemented a Curriculum Learning algorithm. The idea behind it was to first train models on datasets with manually balanced data distribution, and then, with more epochs, to increase the number of tokens that were not part of any coupon. Last epochs would be performed on the original dataset.

   At the end, this approach was also abandoned. Similarly, to the xml preservation, it was performing worse than the basic approach (see plots A.1, B.2, A.3; models with "no-curr" are the ones fine-tuned without the curriculum learning). Additionally, later data batches from Murmuras were free from that flaw. Most of the timestamps had around 30 % - 50 % or more of words that were a part of some coupon.

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, with the model without XML tree preservation and Curriculum learning reached 97 % precision and almost 99 % recall (see plots A.1, A.2 and A.3) 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.

For per-application fine-tuning, models generally performed poorly. Almost all models didn't suprass 20 % in recall metric. An exception was the model trained on the Lidl dataset, with over 60% recall (see plot A.4).

In both incremental fine-tuning scenarios, although the model initially trained on DM data struggled with classification (almost 0 % recall), further fine-tuning significantly improved its recall, with best result for separate fine-tunings being 72 % for Rossmann fine-tuning, and 87 % in case of combined fine-tuning for also the Rossman fine-tuning. This suggests that it is possible to extend the model's functionality with minimal effort through simple fine-tuning (see plots A.5 and A.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 (see plots B.1, B.2, B.3).

The single-application experiment produced results similar to those observed in the selection pass. Once again, the model fine-tuned on Lidl dataset performed best, with 64 % recall (see plot B.4).

For both types of incremental experiments, the results closely mirrored those of the selection pass. For separate fine-tuning, Rossmann fine-tuning reached 95 % recall, and for combined one, 99 % recall. 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 (see plots B.5, B.6).

Chapter 7

Fine-tuned Llama model

Apart from the aforementioned BERT-based proposal, we have prepared a fine-tuned version of Llama[82] model to measure its performance in the studied task as a scrapping tool. The following sections focus on our setup for this proposal and the achieved results.

7.1. The choice of the model

Our primary model selection criterion was the model size - we wanted it to be deployable on as many smartphones as possible. That is why we have rejected models with more than 1 billion parameters, like *Gemma 2 2B*[116] or *Qwen2 1.5B*[117]. We have also decided not to use models with acceptable size, but insufficient performance, like the 360M variant of *SmolLM2*[118]. *GPT-Neo*[119] was disqualified due to the lack of support from the Unsloth tool. Finally, we have chosen the 1B variant of the already mentioned *llama3.2*.

7.2. Task setup

We assumed that the model is expected to take the content of the text fields displayed to the user on the screen and return a list of coupons in JSON format described in section 1.5. Additionally, we have explored the model's performance when the prompt was enhanced with the task description. However, no significant gain was achieved by us this way. We will refer to the llama fine-tuned with the task description as *llama-w* and to the llama fine-tuned without it as *llama-wth*. The full description of the prompt format is given in4.2.2.

7.3. Fine-Tuning

The fine-tuning was performed on the Modal platform with the help of the Unsloth tool. We used the cross-entropy loss[115] and the AdamW optimizer adapted for 8-bit cached values[114]. Finally, we have applied weight decay and used a linear learning rate scheduler. The split between training and evaluation samples in the datasets was 80:20.

7.4. Results of the fine-tuning

In this part, we will analyze only cross-entropy loss evaluated during the training here as the models' performance on our benchmark is described in section 8.4. The detailed plots for fine-tuning results are available in Appendix F.

7.4.1. Baseline fine-tuning

The baseline fine-tuning was performed on the datasets from four apps: REWE, ROSSMANN, LIDL and DM. During the training, the test loss was computed on the datasets from PENNY and EDEKA applications.

During the training of llama-wth, we observe the constant decrease of the training loss. It reaches the value below 0.2 after 9 epochs (see plot F.1). The evaluation loss drops to the value around 0.7 after initial epochs and then grows slowly to reach a value around 0.8 after 10 epochs. The test loss reached a value close to 1.3 after the first epoch and continued to grow over the rest of the epochs. This suggests that the model is overfitting to the training data. The llama-w training plot (see F.2) shows us that the test loss is at approximately 1.25 after the first epoch, the evaluation loss is close to 0.6 in its minimum and the train loss value after 10 epochs is around 0.1. This is a significant improvement compared to llama-wth, however, the overfitting still occurs.

7.4.2. Application-wise fine-tuning

We have also tested the model's ability to generalize when the fine-tuning was performed on the dataset consisting of only one application. In this case, we have measured the training and evaluation loss on the data from the selected application, while the test loss was computed on the EDEKA and PENNY applications, as in the baseline experiment. Each experiment came in a variant with the task description inserted into the prompt and without it.

When training on the prompt without the description, the test loss was significantly higher than in case of training on 4 applications simultaneously (see plot F.3). After the first epoch it reached a value around 2.95 in case of the dataset from ROSSMANN, between 2.8 and 2.9 in case of DM and REWE, and a value slightly lower than 1.8 in case of LIDL. It continued to grow in the following epochs. However, when the training set contained the task description, the test loss for ROSSMANN after the first epoch was on the level comparable to experiments from the previous subsection and it dropped further to a value around 1.15 after several epochs. After that, it started to grow. In case of LIDL, REWE and DM, the test loss reached its minimum after the first epoch with value 1.15 (see plot F.4).

7.4.3. Experiments

Additionally, we have performed experiments similar to the ones in Section 6.3.1. We have performed the incremental combined fine-tuning where every K epochs we introduced a new application to the dataset. Additionally, we have conducted an incremental separate fine-tuning where the dataset was replaced each K epochs. We will refer to the newly introduced portion of the data as *increment*. The results are described in the following subsubsections.

Incremental combined fine-tuning

We have tested different sizes of increments: 15, 50 and 100 (see plots F.5 F.7 F.9). The evaluation loss was computed on the same applications as in the train set. When the task description is missing, we see that in each case the evaluation loss stabilizes around 0.9. However, it is worth noticing that when increment size is equal to 15, the evaluation loss drops to a value around 0.5 and then increases while in case of the greater values it starts at higher level and is decreasing as the training progresses. The test loss seems to fluctuate between 1.2 and 1.3 in the case of an increment size equal to 15 (see F.14), behave in a more

stable way around 1.2 for size 50 (see F.15) and stabilize around 1.2 when increment size is 100 (see F.16).

When training with the task description the test loss fluctuates around 1.15, even for an increment of size 15, which suggests that the amount of the data required to fine-tune existing model to the new application might be small (see plots F.17 F.18 F.19).

Fine-tuning with dataset replacements

In this experiment, the evaluation loss was computed on the data from all apps used in the previous increments. When training without the task description, the evaluation loss plots look similar regardless of increment size (see plots F.11 F.12 F.13). The evaluation loss is at first around 0.5, then it jumps to around 0.85, finally, after the last dataset substitution, it ends around 0.9. These visualizations allow us to suggest the hypothesis that the model *forgets* the overfitted dataset-specific knowledge as it no longer sees the samples from this dataset. The test loss jumps to the value above 1.4 after the first dataset swap and then drops to the value around 1.15 (see plots F.14 F.15 F.16). This suggests that this methodology might be considered an interesting alternative for standard fine-tuning.

Addition of the task description further decreases the test loss, to values below 1.1 in later epochs for an increment size of 10 (see F.20), and even reached 1.05 in the case of larger increment sizes (see F.21 and F.22).

Chapter 8

Benchmark and results

The objective of our custom 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.

By leveraging the benchmark and conducting a series of additional evaluations, we were able to systematically identify the pipeline that delivered the most satisfactory overall performance.

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 C). 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 number 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 instance, during the parsing of the pipeline's JSON output, any cases of incorrectly formatted data or coupons containing unexpected fields are 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.

In the figures presented in Appendix D, as well as in Figures E.1 and E.2, the models are evaluated across six datasets. The Penny and Edeka datasets were used in their entirety, while for DM, Lidl, Rewe, and Rossmann, only the test splits were used.

For a detailed description of the datasets, see Chapter 4. Information on the BERT-based pipelines is provided in Chapter 6, while the Llama-based pipelines are discussed in Chapter 7.

8.5. Pipeline comparison

Based on Figures D.1 and D.2, BERT-top_score emerged as the best-performing variant within the BERT-based pipelines. It consistently outperformed BERT-first in both recall and precision. While the comparison with BERT-concat is less clear-cut, BERT-top_score achieved more than twice the recall on the DM dataset, which was a decisive factor in its overall evaluation.

Similarly, Figures D.3 and D.4 show that Llama-wth outperforms Llama-w. While Llama-w demonstrated slightly better precision, recall was prioritized as the more important metric, as previously discussed.

When comparing BERT-top_score to Llama-wth (Figures D.5 and D.6), the latter clearly emerges as the superior approach in terms of output quality.

Overall, the pipelines performed significantly better on the datasets they were trained on—namely DM, Lidl, Rewe, and Rossmann—compared to the previously unseen datasets, Penny and Edeka. However, Figures D.5 and D.6 suggest that the Llama-based pipelines exhibit a higher potential for generalization than their BERT-based counterparts.

8.6. Llama quantization comparison

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 (see Chapter 5) 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 E.1, E.2, E.3, and E.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

In this thesis, we have introduced two approaches for structured content retrieval from mobile device screen content.

To measure the alignment of the models' outputs with the learning target we have designed a benchmark that compares the sets of coupons using a similarity metric between strings computed on coupons' attributes.

We have fine-tuned a `Llama` model and tested it on a smartphone. We have tested fine-tuning with and without the description of the task. The results of the experiments suggest that the pipeline retrieves most of the coupons. The model fine-tuned without the task description performs better than its counterpart with it. However, the generalization to unseen applications remains an open problem.

We have presented our second proposal, the pipeline consisting of two fine-tuned `BERT` models. We tested vanilla fine-tuning, fine-tuning with `JSON`-encoded content, and an approach inspired by curriculum learning. We have also measured the performance of different token combining policies in the extraction pass. The training process without enhancements proved to be the most efficient approach, resulting in a pipeline that performs nearly as well as the `Llama`-based one when combined with the `top-score` token combination rule.

9.1. Possible extensions

9.1.1. The `Llama`'s input in `JSON` format

We have tried providing the `BERT` model the content of the text fields organized into a `JSON` tree to preserve the information about the layout. This did not improve the model's performance. However, it is possible that a larger model could take advantage of this information and yield better results. More specifically, organizing the `Llama` pipeline input this way during training could produce interesting results. The challenge here would be to keep the layout processing speed at a reasonable rate as wrapping text fields into `JSON` tree would require longer prompts.

9.1.2. Benchmarking with the help of language models

Currently, our benchmarking procedure relies on a deterministic algorithm that matches coupons if their attributes have similar spelling. This approach has a weakness in case of benchmarking a generative pipeline, especially our `Llama`-based proposal. If the model's output is semantically correct but differs in spelling, i.e. the returned validity date is in a different format, the benchmark can possibly treat this as complete or a near-complete mismatch (see

??). This problem could be addressed by employing LLM-as-a-judge and skilfully prompting a significantly more powerful language model than `Llama`.

9.1.3. Full Mobile Deployment

Although the performance of the `Llama` model has been evaluated on a mobile device, the development of a production-ready mobile application based on our proposed pipelines remains outside the scope of this thesis. A potential deployment of the `Llama` model could utilize the `llama.cpp` framework, whereas the fine-tuned `BERT` model is compatible with the `ExecuTorch` framework [131]. Such deployment would also necessitate integration with the Company's screen content retrieval tools and the web API responsible for data collection. An unresolved design consideration is whether inference should be executed in real time or deferred to periods of user inactivity, such as nighttime. This decision should be guided by model performance characteristics: the `Llama` model currently does not meet the desired token processing speed [132], while the performance of the `BERT`-based pipeline remains to be fully assessed.

Appendix A

BERT selection pass fine-tuning results

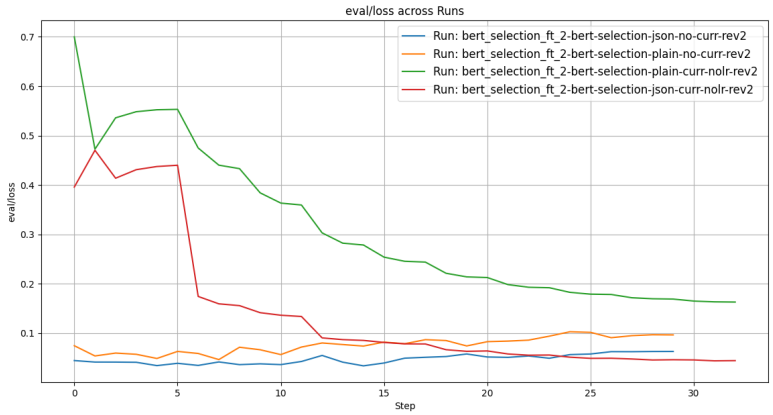

Figure A.1: Eval/loss statistics of selection pass during general training

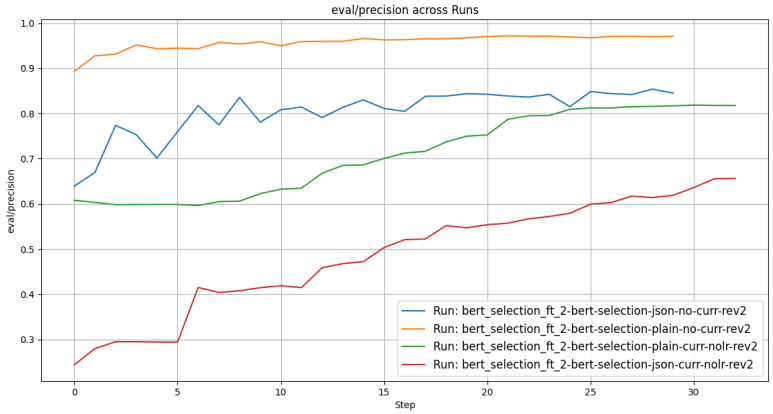

Figure A.2: Eval/precision statistics of selection pass during general training

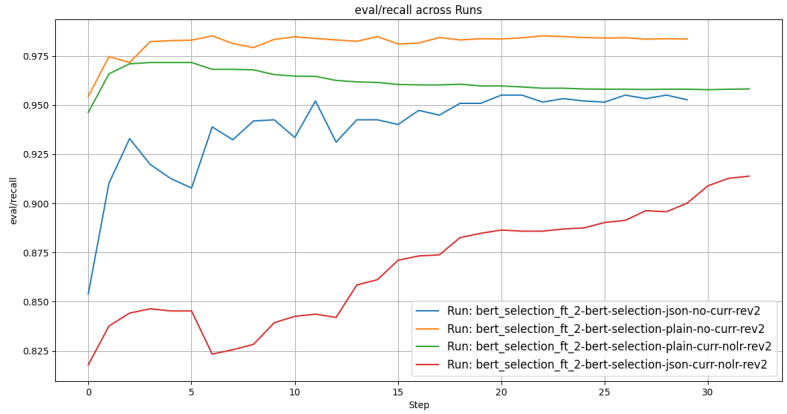

Figure A.3: Eval/recall statistics of selection pass during general training

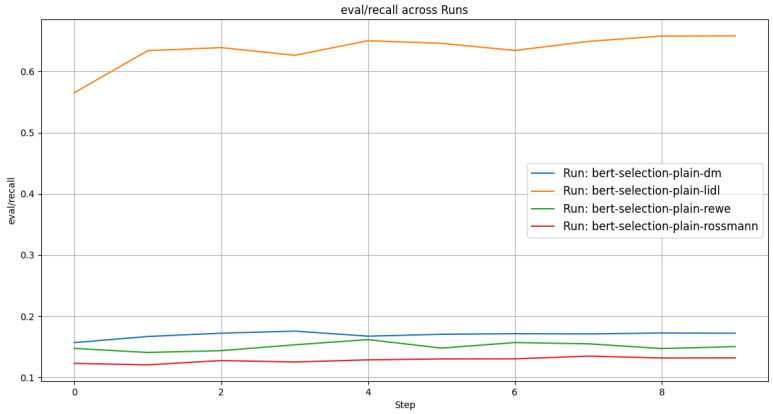

Figure A.4: Eval/recall statistics of selection pass during training on separate apps

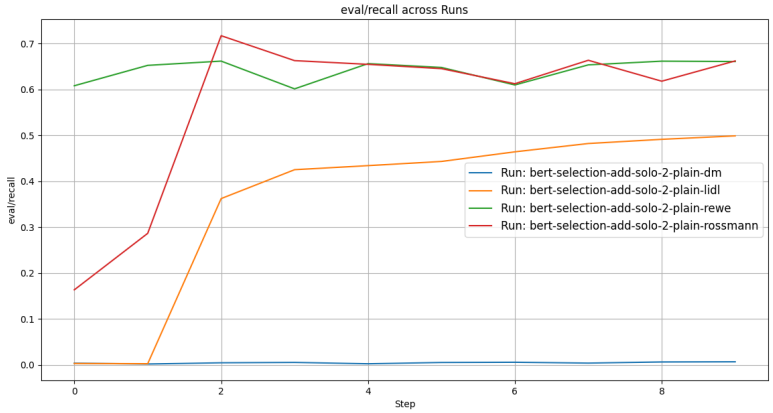

Figure A.5: Eval/recall statistics of selection pass during incremental separate training

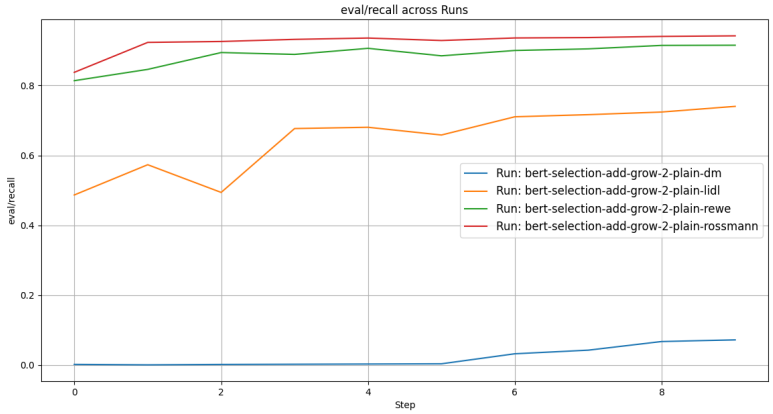

Figure A.6: Eval/recall statistics of selection pass during incremental combined training

Appendix B

BERT extraction pass fine-tuning results

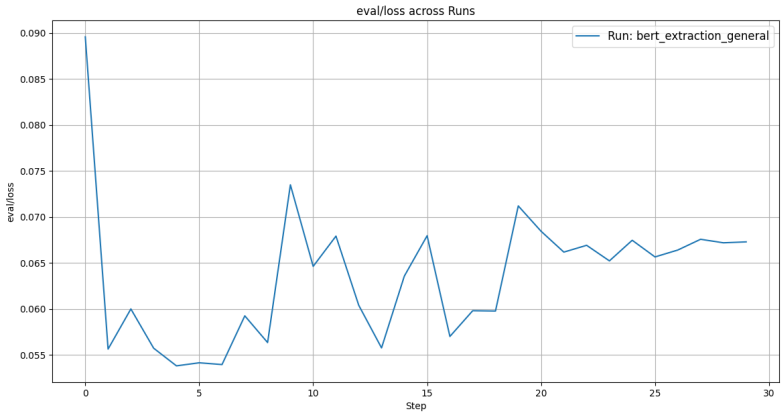

Figure B.1: Eval/loss statistics of extraction pass during general training

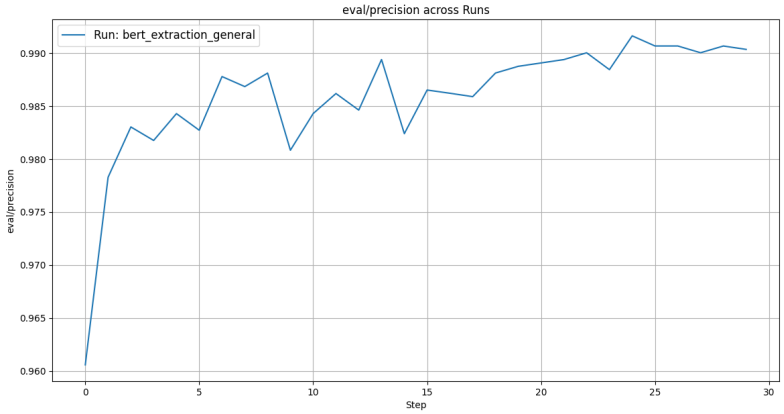

Figure B.2: Eval/precision statistics of extraction pass during general training

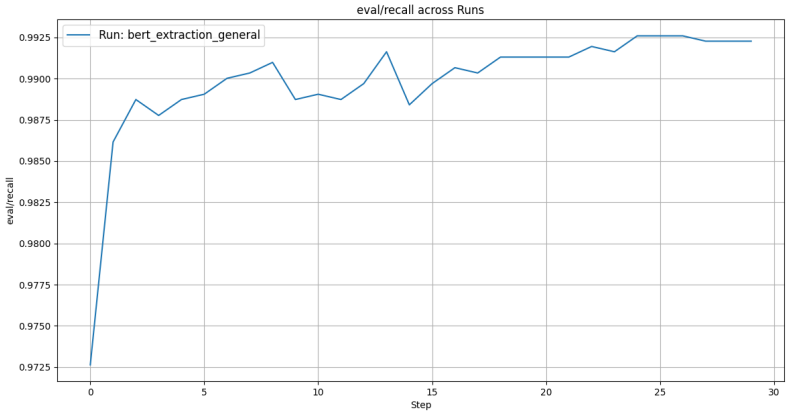

Figure B.3: Eval/recall statistics of extraction pass during general training

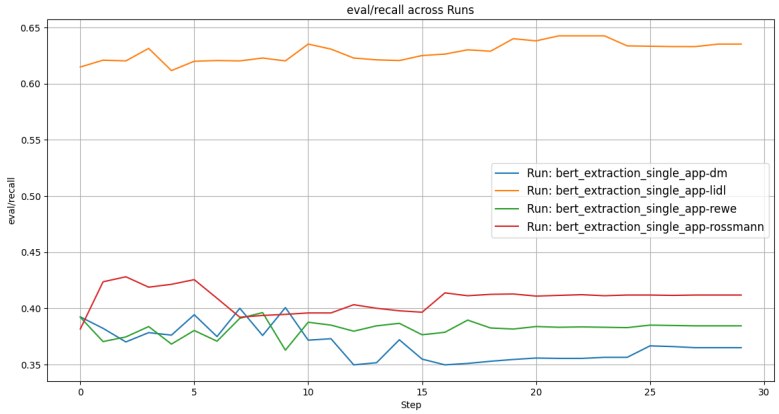

Figure B.4: Eval/recall statistics of extraction pass during separate training

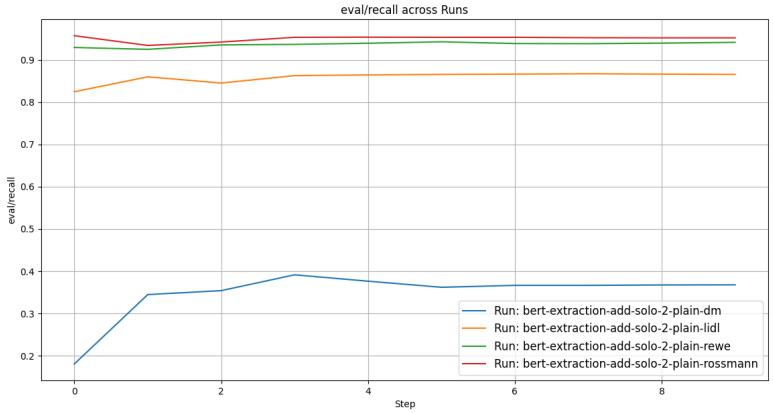

Figure B.5: Eval/recall statistics of extraction pass during incremental separate training

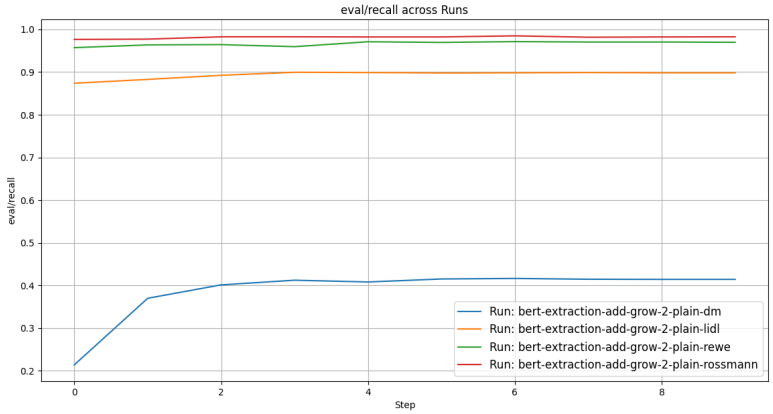

Figure B.6: Eval/recall statistics of extraction pass during incremental combined training

Appendix C

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`.

C.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.

C.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. In case the sum of remaining fields' weights is 0, the similarity of the coupons is set to 1.

Appendix D

Pipeline comparison plots

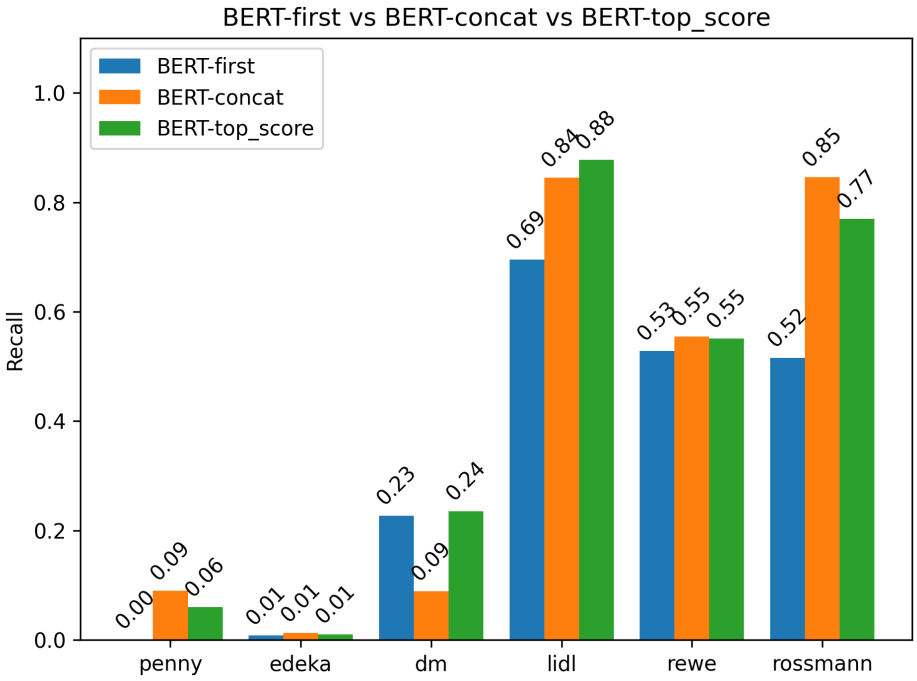

Figure D.1: BERT-first vs BERT-concat vs BERT-top_score - recall

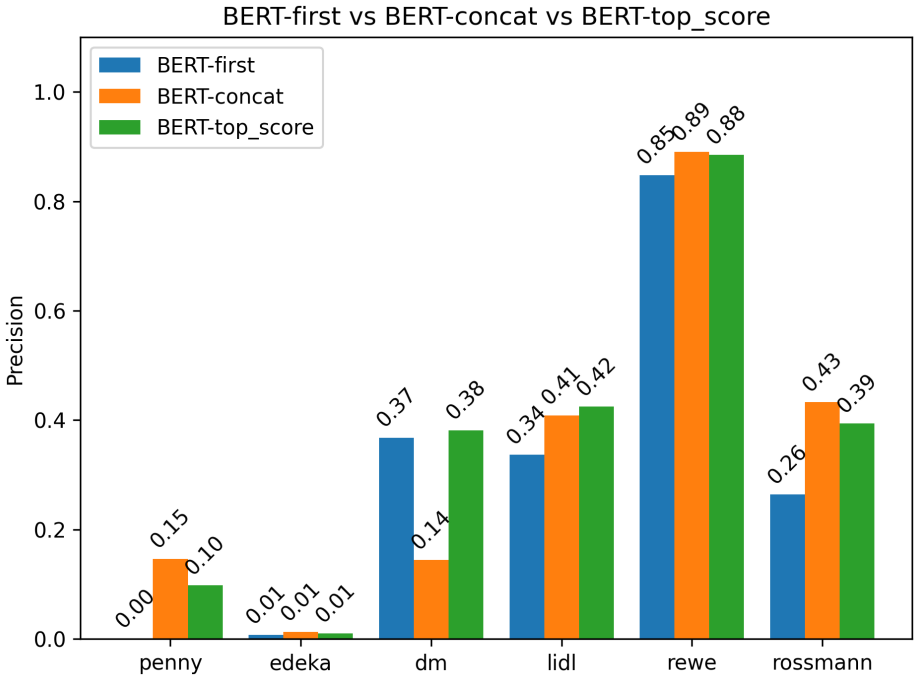

Figure D.2: BERT-first vs BERT-concat vs BERT-top_score - precision

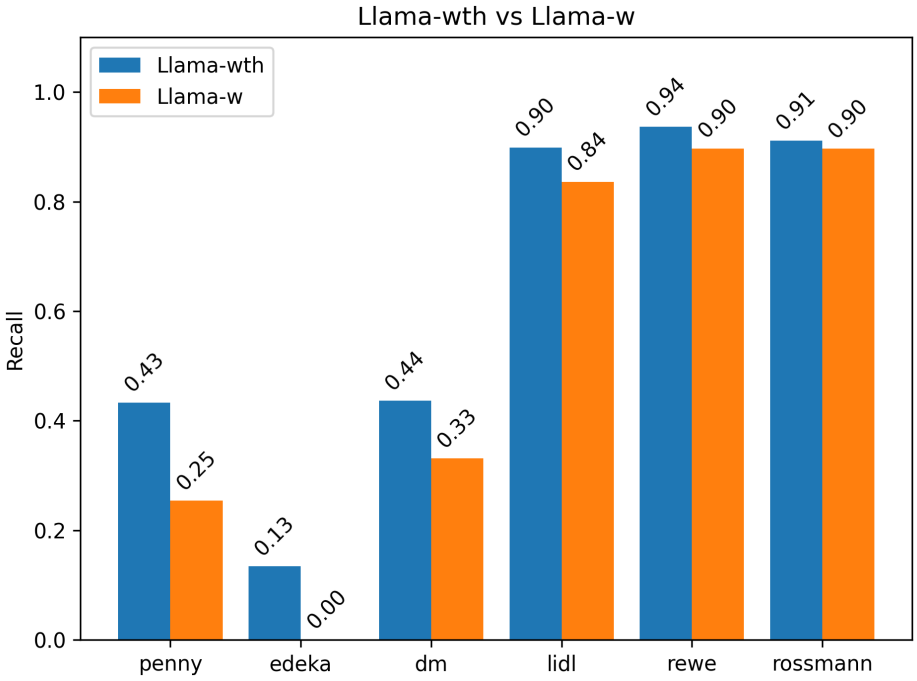

Figure D.3: Llama-wth vs Llama-w - recall

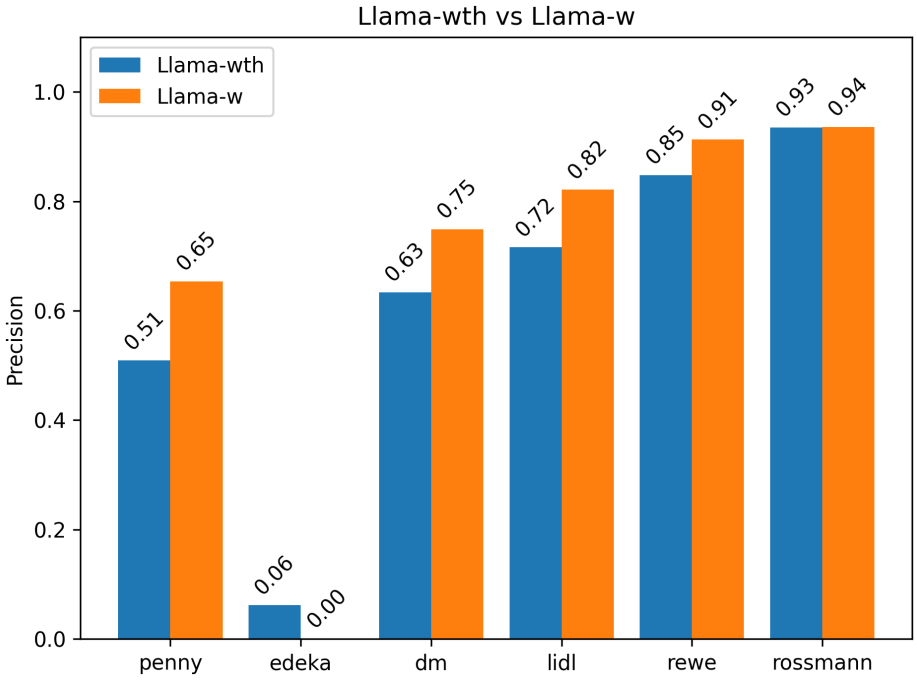

Figure D.4: Llama-wth vs Llama-w - precision

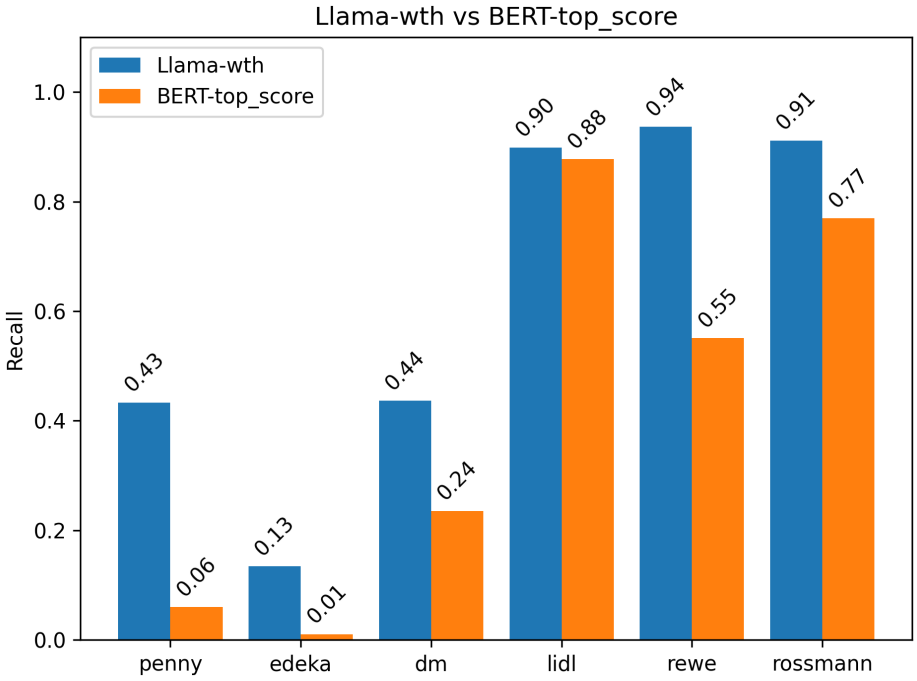

Figure D.5: Llama-wth vs BERT-concat - recall

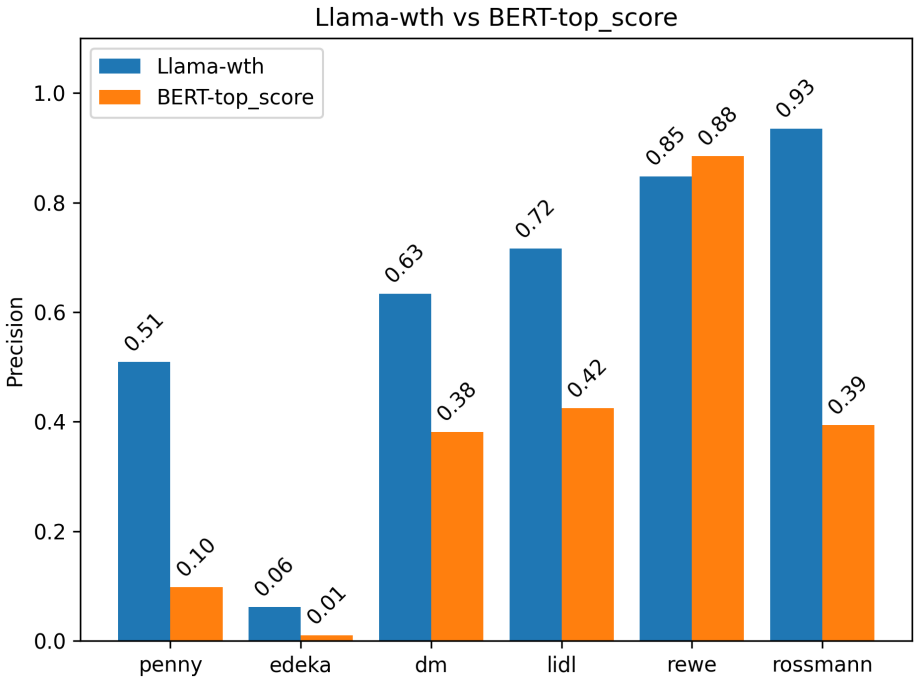

Figure D.6: Llama-wth vs BERT-concat - precision

Appendix E

Llama Quantization Comparison Plots

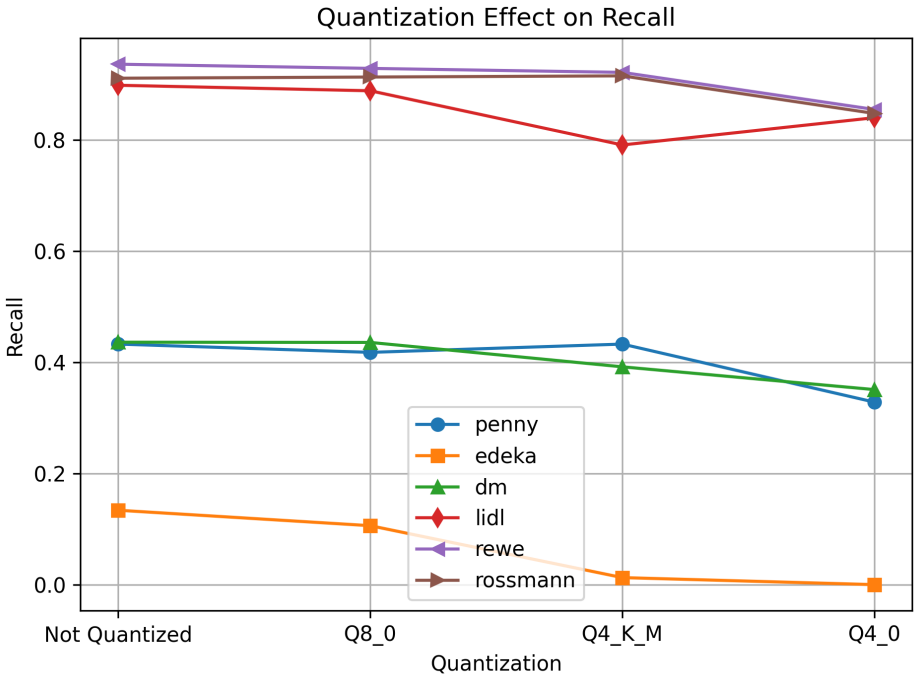

Figure E.1: Llama-wth quantization effect on recall

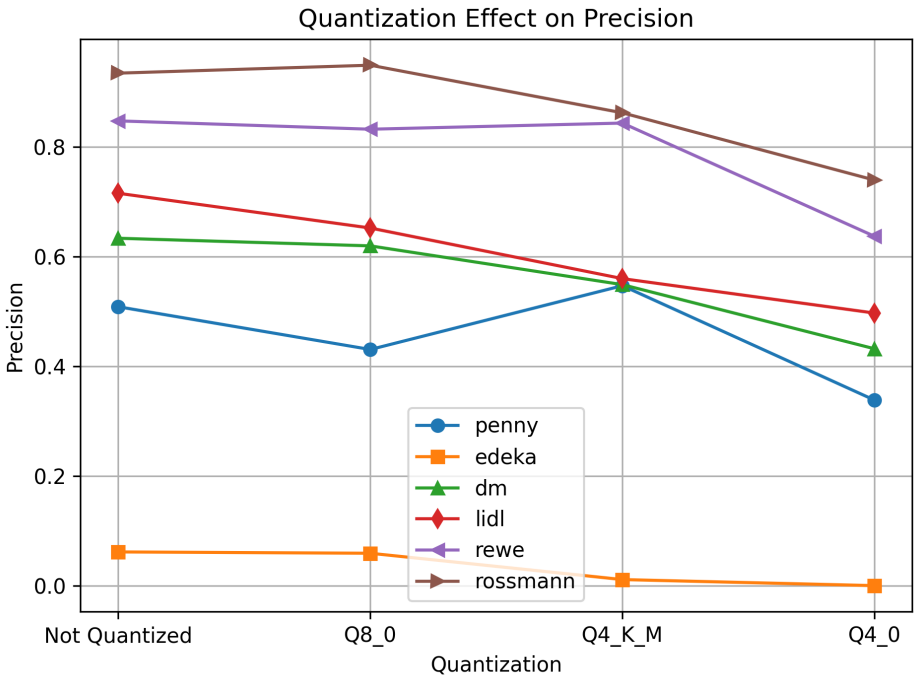

Figure E.2: Llama-wth quantization effect on precision

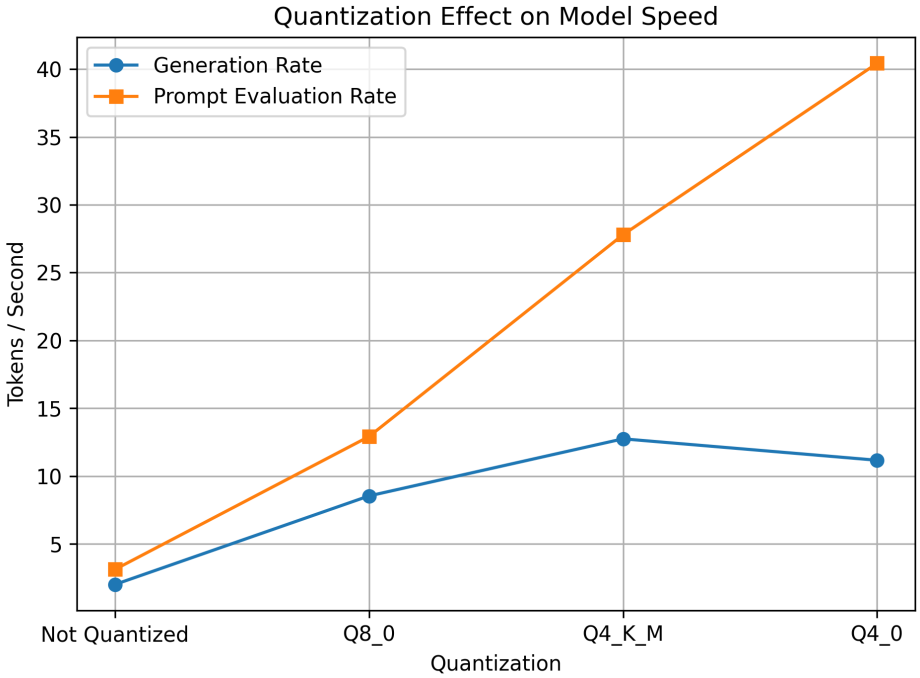

Figure E.3: Llama-wth quantization effect on model speed

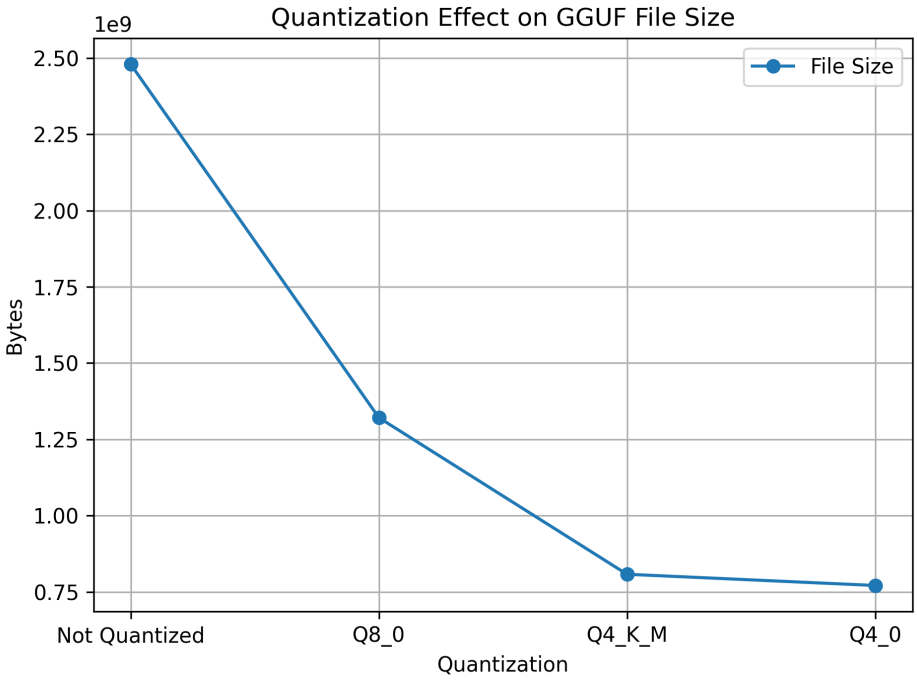

Figure E.4: Llama-wth quantization effect on GGUF file size

Appendix F

Llama fine-tuning plots

Note that the epoch numbering starts from 0. That means the leftmost value on the plots is the value after the first epoch. In case of incremental fine-tunings, the series name indicates the source of the data in increment that was added recently.

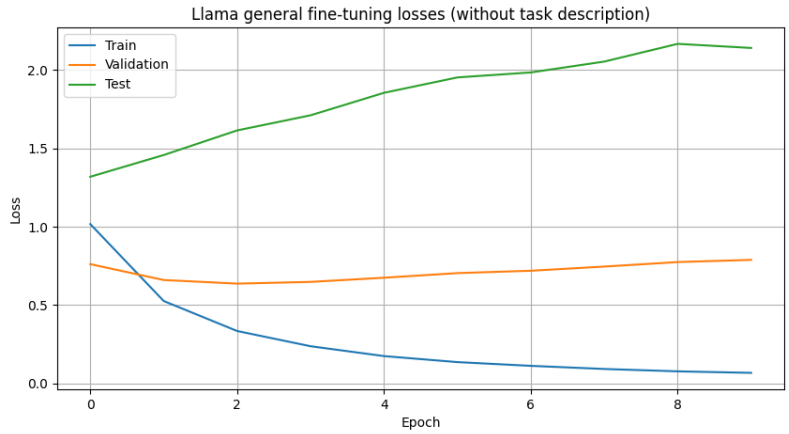

Figure F.1: Llama fine-tuning losses for baseline experiment.

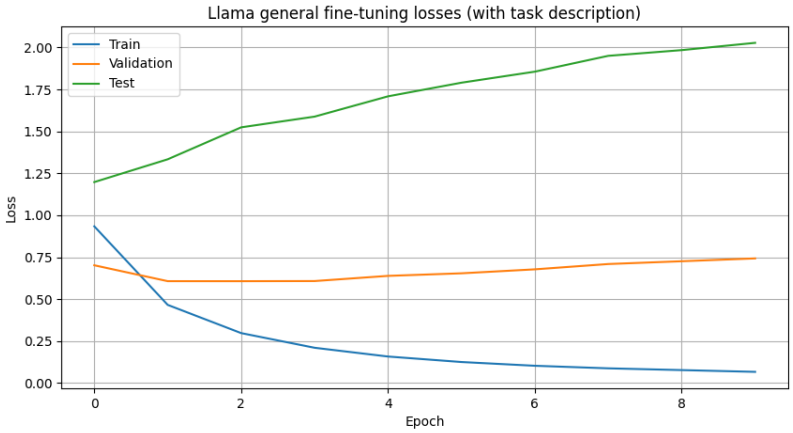

Figure F.2: Llama fine-tuning losses for baseline experiment (with task description).

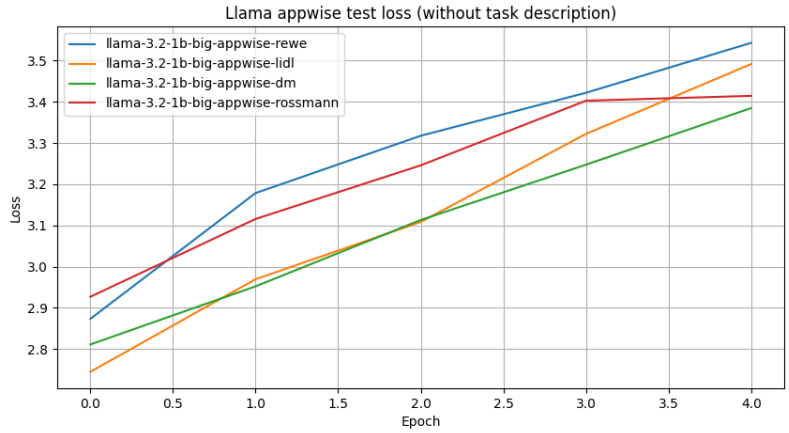

Figure F.3: Llama fine-tuning test losses for application-wise runs.

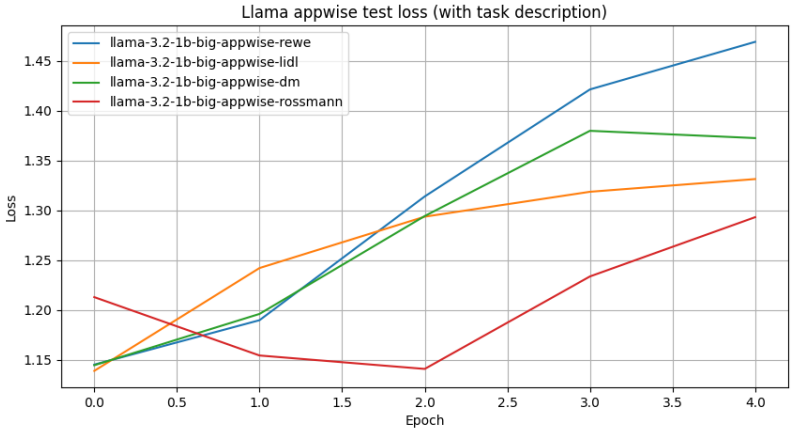

Figure F.4: Llama fine-tuning test losses for application-wise runs (with task description).

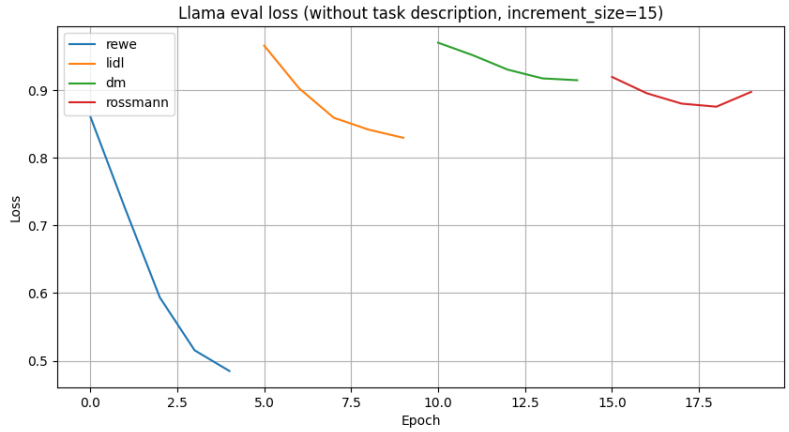

Figure F.5: Llama evaluation loss for fine-tuning with concatenated increments of size 15.

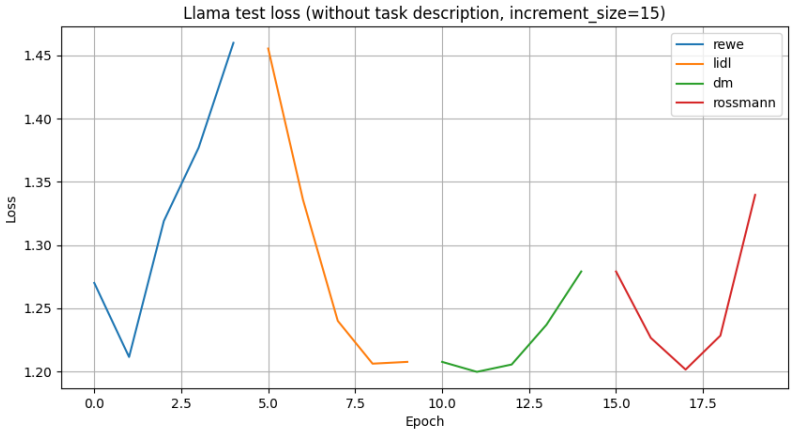

Figure F.6: Llama test loss for fine-tuning with concatenated increments of size 15.

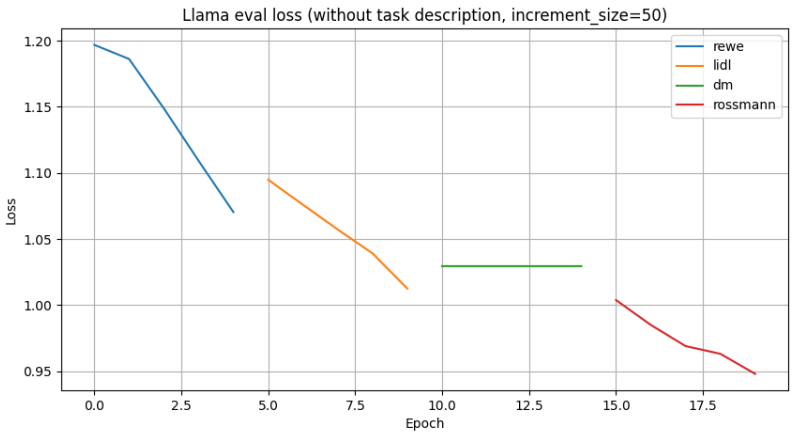

Figure F.7: Llama evaluation loss for fine-tuning with concatenated increments of size 50.

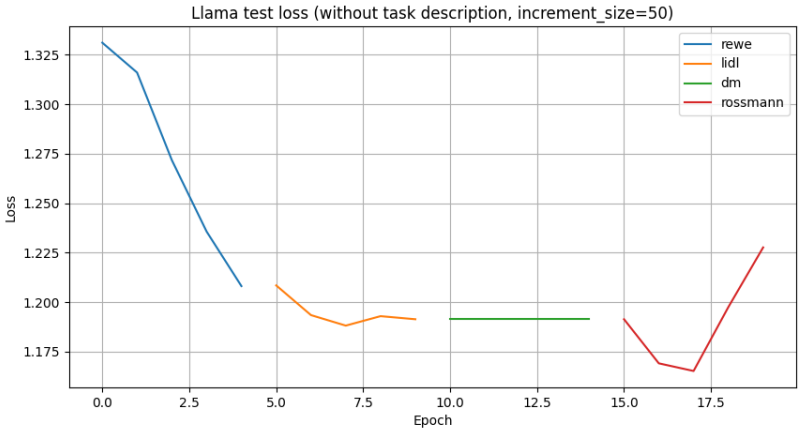

Figure F.8: Llama test loss for fine-tuning with concatenated increments of size 50.

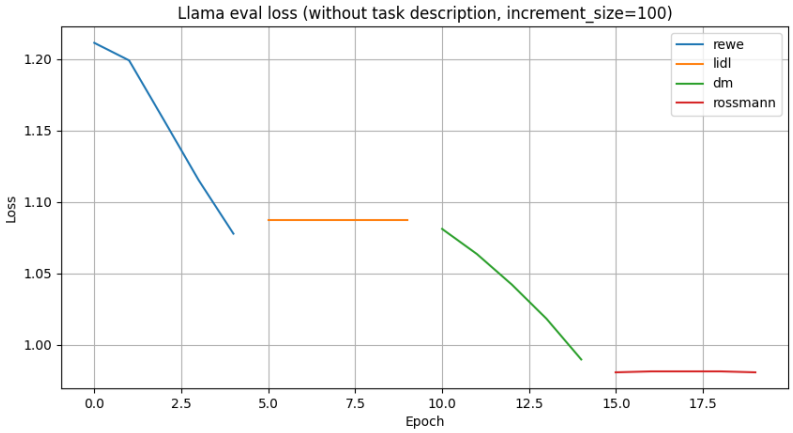

Figure F.9: Llama evaluation loss for fine-tuning with concatenated increments of size 100.

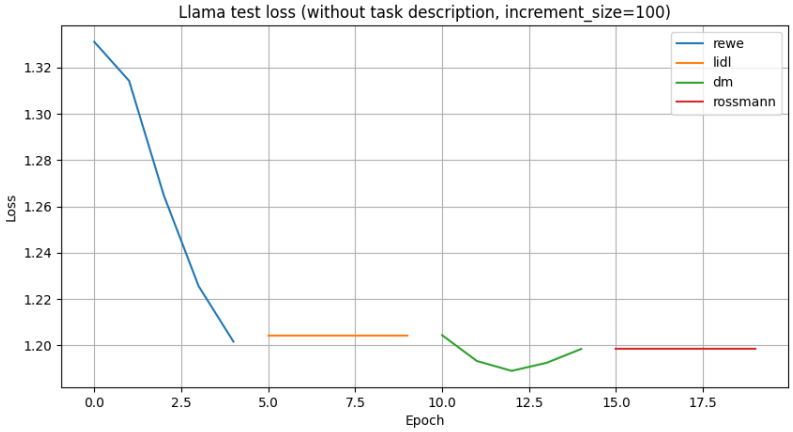

Figure F.10: Llama test loss for fine-tuning with concatenated increments of size 100.

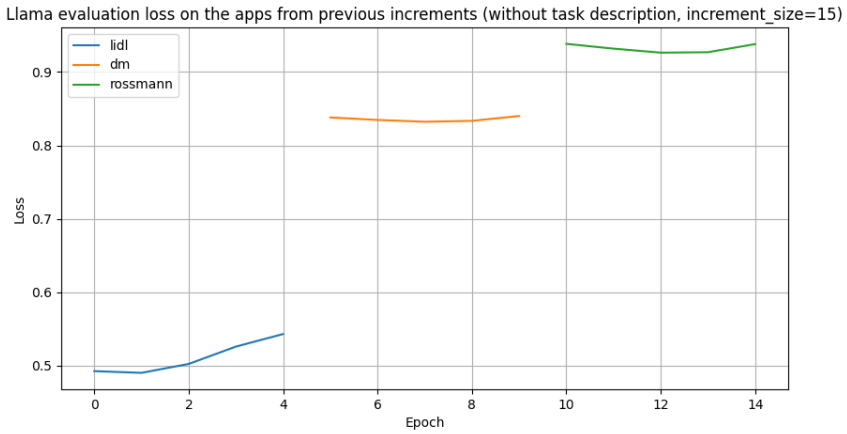

Figure F.11: Llama evaluation loss for previous increments for fine-tuning with increments of size 15.

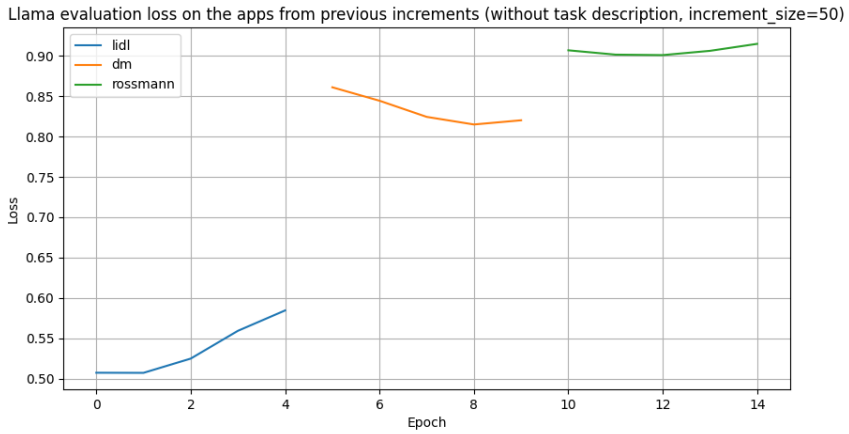

Figure F.12: Llama evaluation loss for previous increments for fine-tuning with increments of size 50.

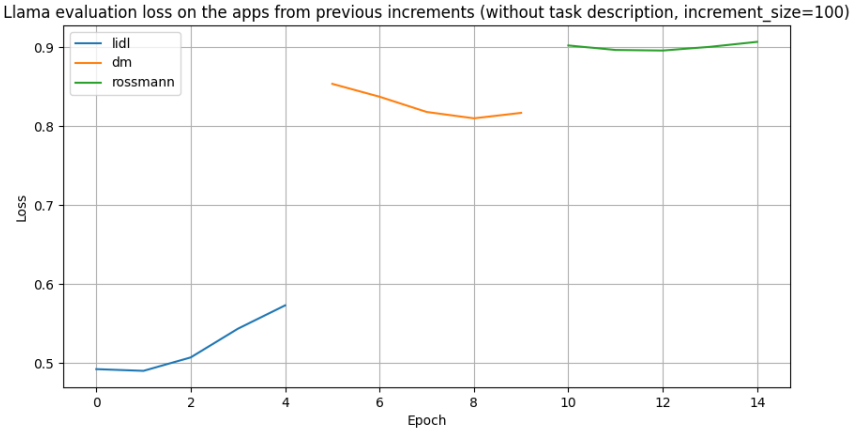

Figure F.13: Llama evaluation loss for previous increments for fine-tuning with increments of size 100.

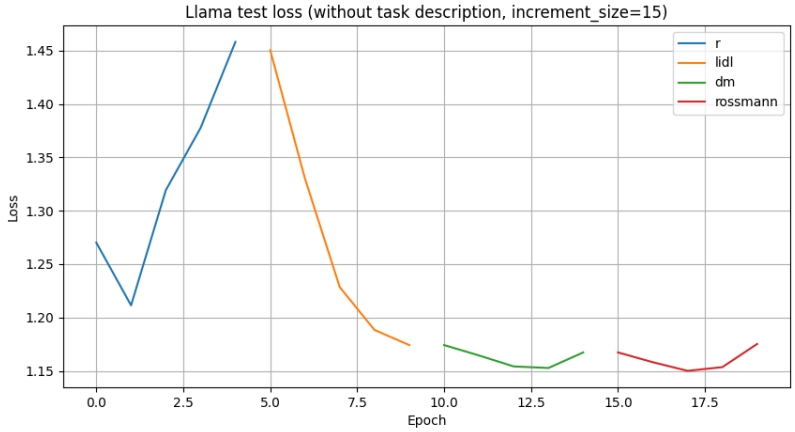

Figure F.14: Llama test loss for fine-tuning with increments of size 15.

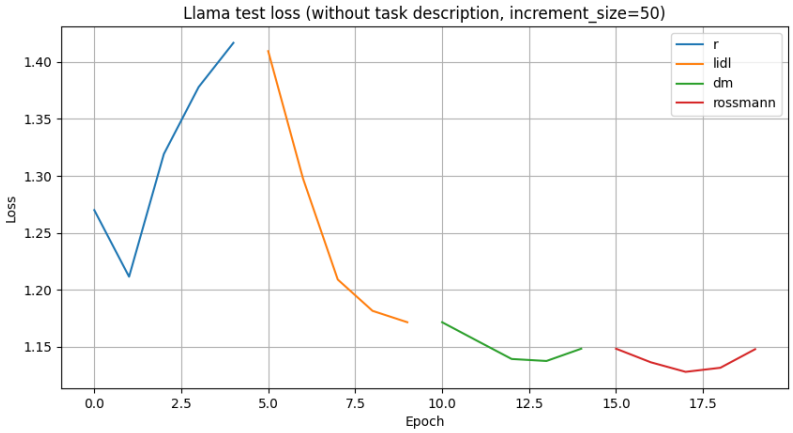

Figure F.15: Llama test loss for fine-tuning with increments of size 50.

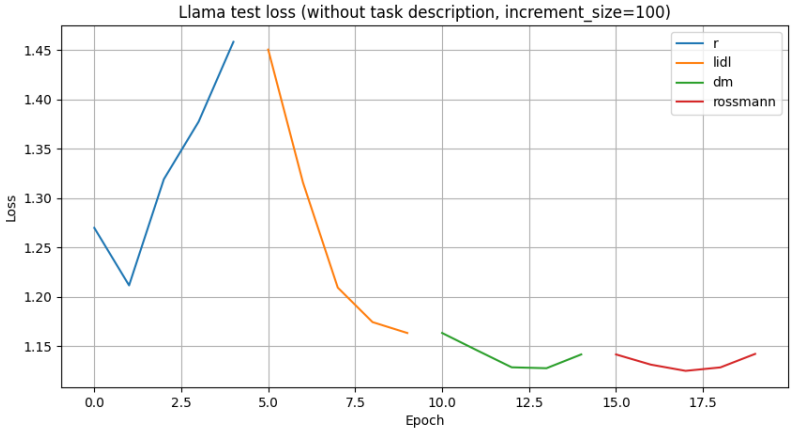

Figure F.16: Llama test loss for fine-tuning with increments of size 100.

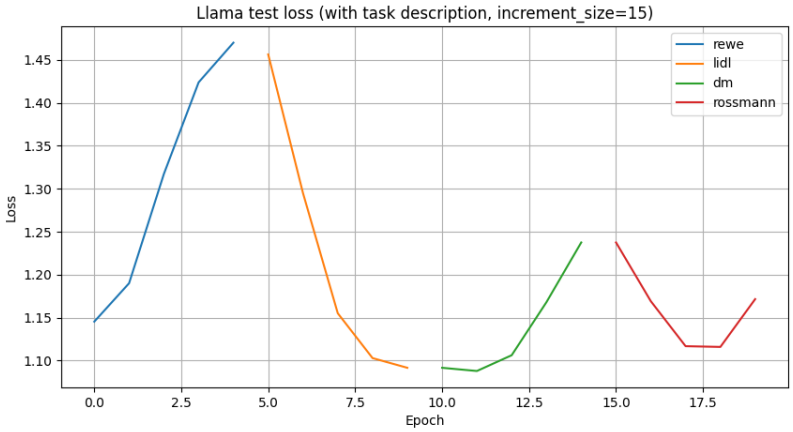

Figure F.17: Llama test loss for fine-tuning with concatenated increments of size 15.

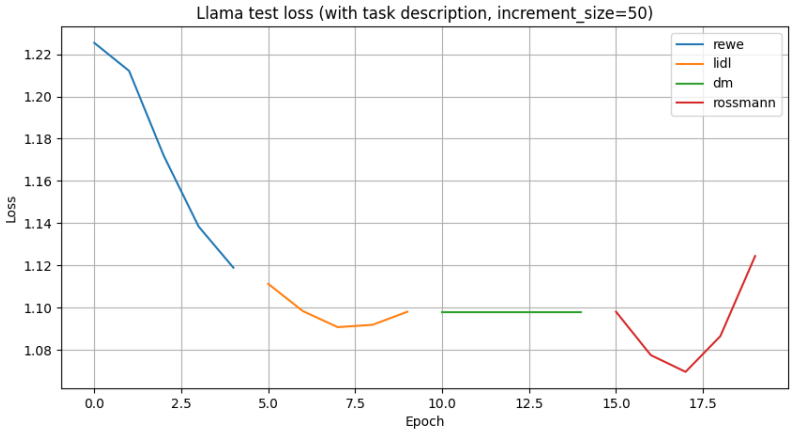

Figure F.18: Llama test loss for fine-tuning with concatenated increments of size 50.

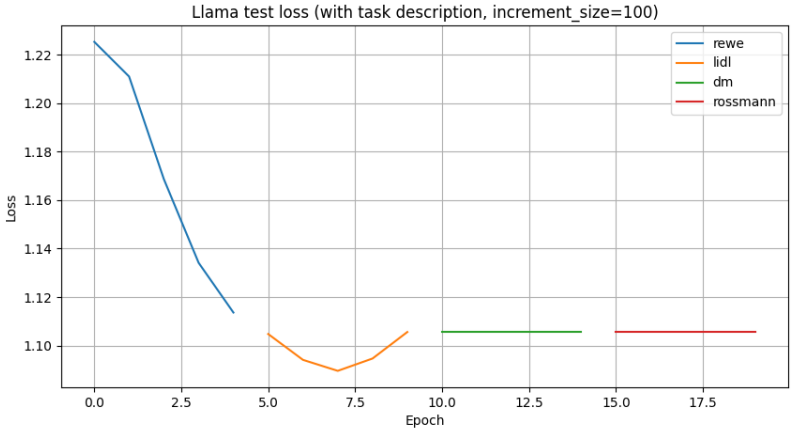

Figure F.19: Llama test loss for fine-tuning with concatenated increments of size 100.

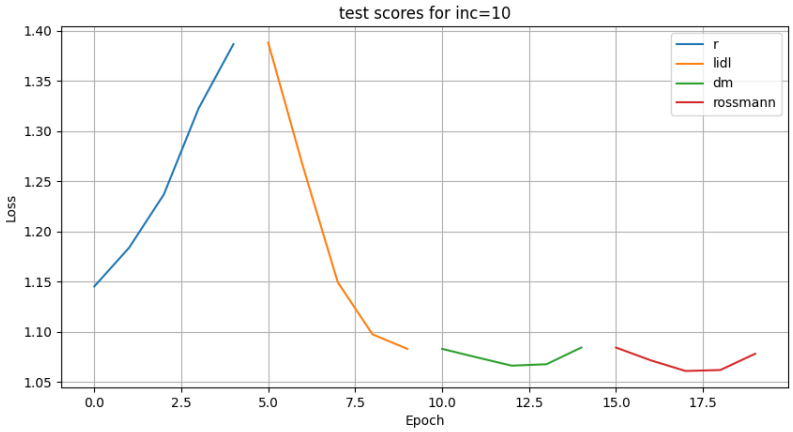

Figure F.20: Llama test loss for fine-tuning with increments of size 10.

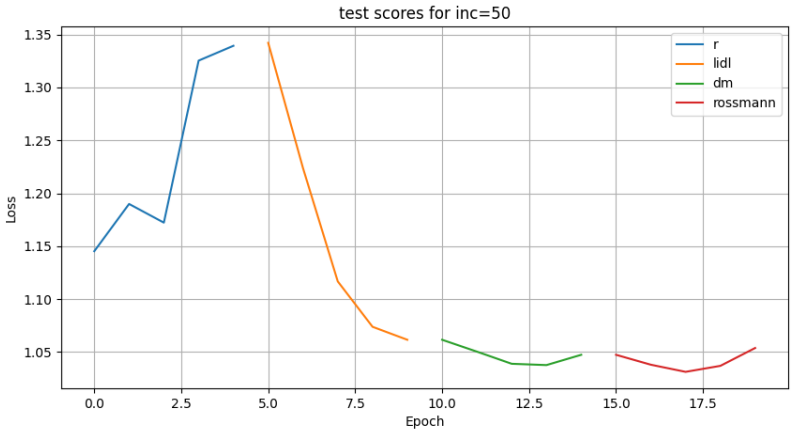

Figure F.21: Llama test loss for fine-tuning with increments of size 50.

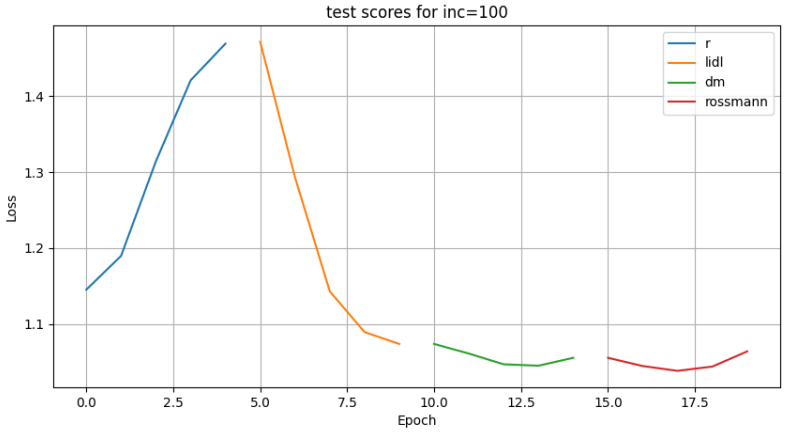

Figure F.22: Llama test loss for fine-tuning with increments of size 100.

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 fine-tuning 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

[112] *Llama 3.2 introduced by Meta; Comparison of llama-3.2-1b with Gemma 2 2B IT* https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/ [Accessed 04.05.2025]

[113] *The list of LLMs supported by the Unsloth tool* https://docs.unsloth.ai/get-started/all-our-models [Accessed 04.05.2025]

[114] *Documentation of the bitsandbytes library provided by the Hugging Face* https://huggingface.co/docs/bitsandbytes/main/en/index [Accessed 04.05.2025]

[115] *Anqi Mao and Mehryar Mohri and Yutao Zhong (2023): Cross-Entropy Loss Functions: Theoretical Analysis and Applications* https://arxiv.org/abs/2304.07288

[116] *Gemma Team and Morgane Riviere and Shreya Pathak and Pier Giuseppe Sessa and Cassidy Hardin and Surya Bhupatiraju and Léonard Hussenot and Thomas Mesnard and Bobak Shahriari and Alexandre Ramé and Johan Ferret and Peter Liu and Pouya Tafti and Abe Friesen and Michelle Casbon and Sabela Ramos and Ravin Kumar and Charline Le Lan and Sammy Jerome and Anton Tsitsulin and Nino Vieillard and Piotr Stanczyk and Sertan Girgin and Nikola Momchev and Matt Hoffman and Shantanu Thakoor and Jean-Bastien Grill and Behnam Neyshabur and Olivier Bachem and Alanna Walton and Aliaksei Severyn and Alicia Parrish and Aliya Ahmad and Allen Hutchison and Alvin Abdagic and Amanda Carl and Amy Shen and Andy Brock and Andy Coenen and Anthony Laforge and Antonia Paterson and Ben Bastian and Bilal Piot and Bo Wu and Brandon Royal and Charlie Chen and Chintu Kumar and Chris Perry and Chris Welty and Christopher A. Choquette-Choo and Danila Sinopalnikov and David Weinberger and Dimple Vijaykumar and Dominika Rogozińska and Dustin Herbison and Elisa Bandy and Emma Wang and Eric Noland and Erica Moreira and Evan Senter and Evgenii Eltyshev and Francesco Visin and Gabriel Rasskin and Gary Wei and Glenn Cameron and Gus Martins and Hadi Hashemi and Hanna Klimczak-Plucińska and Harleen Batra and Harsh Dhand and Ivan Nardini and Jacinda Mein and Jack Zhou and James Svensson and Jeff Stanway and Jetha Chan and Jin Peng Zhou and Joana Carrasqueira and Joana Iljazi and Jocelyn Becker and Joe Fernandez and Joost van Amersfoort and Josh Gordon and Josh Lipschultz and Josh Newlan and Ju-yeong Ji and Kareem Mohamed and Kartikeya Badola and Kat Black and Katie Millican and Keelin McDonell and Kelvin Nguyen and Kiranbir Sodhia and Kish Greene and Lars Lowe Sjoesund and Lauren Usui and Laurent Sifre and Lena*

Heuermann and Leticia Lago and Lilly McNealus and Livio Baldini Soares and Logan Kilpatrick and Lucas Dixon and Luciano Martins and Machel Reid and Manvinder Singh and Mark Iverson and Martin Görner and Mat Velloso and Mateo Wirth and Matt Davidow and Matt Miller and Matthew Rahtz and Matthew Watson and Meg Risdal and Mehran Kazemi and Michael Moynihan and Ming Zhang and Minsuk Kahng and Minwoo Park and Mofi Rahman and Mohit Khatwani and Natalie Dao and Nenshad Bardoliwalla and Nesh Devanathan and Neta Dumai and Nilay Chauhan and Oscar Wahltinez and Pankil Botarda and Parker Barnes and Paul Barham and Paul Michel and Pengchong Jin and Petko Georgiev and Phil Culliton and Pradeep Kuppala and Ramona Comanescu and Ramona Merhej and Reena Jana and Reza Ardeshir Rokni and Rishabh Agarwal and Ryan Mullins and Samaneh Saadat and Sara Mc Carthy and Sarah Cogan and Sarah Perrin and Sébastien M. R. Arnold and Sebastian Krause and Shengyang Dai and Shruti Garg and Shruti Sheth and Sue Ronstrom and Susan Chan and Timothy Jordan and Ting Yu and Tom Eccles and Tom Hennigan and Tomas Kocisky and Tulsee Doshi and Vihan Jain and Vikas Yadav and Vilobh Meshram and Vishal Dharmadhikari and Warren Barkley and Wei Wei and Wenming Ye and Woohyun Han and Woosuk Kwon and Xiang Xu and Zhe Shen and Zhitao Gong and Zichuan Wei and Victor Cotruta and Phoebe Kirk and Anand Rao and Minh Giang and Ludovic Peran and Tris Warkentin and Eli Collins and Joelle Barral and Zoubin Ghahramani and Raia Hadsell and D. Sculley and Jeanine Banks and Anca Dragan and Slav Petrov and Oriol Vinyals and Jeff Dean and Demis Hassabis and Koray Kavukcuoglu and Clement Farabet and Elena Buchatskaya and Sebastian Borgeaud and Noah Fiedel and Armand Joulin and Kathleen Kenealy and Robert Dadashi and Alek Andreev (2024): Gemma 2: Improving Open Language Models at a Practical Size https://arxiv.org/abs/2408.00118

[117] *An Yang and Baosong Yang and Binyuan Hui and Bo Zheng and Bowen Yu and Chang Zhou and Chengpeng Li and Chengyuan Li and Dayiheng Liu and Fei Huang and Guanting Dong and Haoran Wei and Huan Lin and Jialong Tang and Jialin Wang and Jian Yang and Jianhong Tu and Jianwei Zhang and Jianxin Ma and Jianxin Yang and Jin Xu and Jingren Zhou and Jinze Bai and Jinzheng He and Junyang Lin and Kai Dang and Keming Lu and Keqin Chen and Kexin Yang and Mei Li and Mingfeng Xue and Na Ni and Pei Zhang and Peng Wang and Ru Peng and Rui Men and Ruize Gao and Runji Lin and Shijie Wang and Shuai Bai and Sinan Tan and Tianhang Zhu and Tianhao Li and Tianyu Liu and Wenbin Ge and Xiaodong Deng and Xiaohuan Zhou and Xingzhang Ren and Xinyu Zhang and Xipin Wei and Xuancheng Ren and Xuejing Liu and Yang Fan and Yang Yao and Yichang Zhang and Yu Wan and Yunfei Chu and Yuqiong Liu and Zeyu Cui and Zhenru Zhang and Zhifang Guo and Zhihao Fan (2024): Qwen2 Technical Report* https://arxiv.org/abs/2407.10671

[118] *SmolLM2 model on Hugging Face* https://huggingface.co/HuggingFaceTB/SmolLM2-360M-Instruct

[119] *Black, Sid and Leo, Gao and Wang, Phil and Leahy, Connor and Biderman, Stella (2021): GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow* https://doi.org/10.5281/zenodo.5297715

[120] *Text Classification in the LLM Era – Where do we stand?* https://arxiv.org/abs/2502.11830 [Accessed 03.05.2025]

[121] *How To Think About End-To-End Encryption and AI: Training, Processing, Disclosure, and Consent* https://arxiv.org/abs/2412.20231 [Accessed 04.05.2025]

[122] *Principal Component Analyzis* https://en.wikipedia.org/wiki/Principal_component_analysis [Accessed 04.05.2025]

[123] *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network* https://www.sciencedirect.com/science/article/abs/pii/S0167278919305974 [Accessed 04.05.2025]

[124] *IEEE 754* https://en.wikipedia.org/wiki/IEEE_754 [Accessed 04.05.2025]

[125] *Web scraping* https://en.wikipedia.org/wiki/Web_scraping [Accessed 06.05.2025]

[126] *BeautifulSoup4* https://pypi.org/project/beautifulsoup4/ [Accessed 06.05.2025]

[127] *DeepSeek* https://www.deepseek.com/ [Accessed 07.05.2025]

[128] *OpenAI API* https://openai.com/api/ [Accessed 07.05.2025]

[129] *Curriculum learning implementation* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/blob/main/src/bert_fine-tuning/curriculer.py [Accessed 15.05.2025]

[130] *JSON algorithm implementation* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/blob/main/src/bert_dataset_generation/generate_coupon_selection_ds.py [Accessed 15.05.2025]

[131] *The pull request to the Hugging Face repository with unit test proving compatibility of BERT with ExecuTorch* https://github.com/huggingface/transformers/pull/34424 [Accessed 25.05.2025]

[132] *Our experiment for estimation of the token processing rate required for the real-time scree content processing.* https://github.com/ZPP-MURMURAS/ZPP_Murmuras/blob/main/research/speed_reuirements_research/tokens_per_sec.ipynb [Accessed 25.05.2025]