ZPP Murmuras
High Level Design

Gustaw Blachowski Szymon Kozłowski
Natalia Junkiert Kamil Dybek

Introduction

The goal of this project is to develop a universal solution for processing phone screen content, such as social media posts and online advertisements. Additionally, all processing must occur locally on the user's device to protect sensitive data.

This project is being developed at the request of **Murmuras**, a company specializing in providing analytical data for both commercial and scientific purposes.

While the proposed method is expected to be applicable across various domains, our work focuses on the task of gathering discount coupons.

A **discount coupon**, in our case, refers to a digital voucher offered to users through an application published by a supermarket chain. With new applications of this type continuously emerging and existing ones evolving, maintaining a reliable data delivery pipeline using conventional scraping techniques is challenging. Our solution leverages Large Language Models (LLMs) to address this issue.

Beyond this, the developed method will also be applicable to tasks such as profiling users' political views without compromising their privacy, as all data processing occurs locally.

Existing Solutions and Their Limitations

To the best of our knowledge, no publicly available solutions directly address our problem. There have been attempts to use LLMs for web scraping [**?**][**?**], but effectively deploying these solutions on mobile devices remains largely unexplored.

A notable drawback of the language models used in these approaches is their extremely large number of parameters. For instance, GPT-3 has 175 billion parameters [**?**], making it challenging to implement on resource-constrained devices.

Murmuras' Solution

The existing prototype developed by Murmuras processes data in CSV format, which describes the screen's content. This data undergoes basic preprocessing, such as the removal of empty columns, before being converted into JSON format using ChatGPT-4 mini.

However, this solution has two major limitations. First, it does not run locally on mobile devices, as the model is not available for on-device execution. Second, the data is often misinterpreted—for example, volume may be mistakenly treated as the product's price.

Development Opportunities

In theory, these problems can be addressed. Python modules such as *outlines* [**?**] and *LangChain* [**?**] allow for enforcing structured output in language models. The former is compatible with models from *HuggingFace*. Combined with mobile model deployment tools like Llama.cpp [**?**], this would enable local inference on mobile devices.

Specification of the Completed Project

The core objective of this project is to design and implement a functional solution capable of extracting discount coupons from a diverse range of mobile application data. While deploying the solution as a mobile application on an Android smartphone is optional, we require our approach to be theoretically feasible for such deployment. Additionally, our solution is expected to be

generalizable to a broader range of problems, as outlined above. We will provide documentation to facilitate this generalization.

Challenges

Hardware Requirements

Computational Power and RAM

Modern LLMs and data preprocessing algorithms often demand substantial computational resources [**?**]. At the same time, we aim for all processing to be performed locally on the mobile device. The challenge lies in selecting a model that is not overly resource-intensive yet remains effective for our use case. To ensure compatibility with most modern smartphones, we aim to use a model with no more than 1 billion trainable parameters [**?**].

Disk Storage

The application will require several gigabytes of disk space [**?**], as the model weights will be stored locally. However, we anticipate this will not be an issue for users, given the company's business model, where users are incentivized to install the data-gathering application on their phones.

Benchmarking

Algorithm

To evaluate the quality of our solution, we propose the following benchmarking metric. Consider a batch of samples S_i representing smartphone screen content in CSV format. We define a *coupon* as a tuple containing the following fields:

1. *old_price*: Fixed-precision number representing the standard price of the item
2. *new_price*: Fixed-precision number representing the discounted price offered to the customer
3. *name*: String representing the item name

4. *validity*: String representing the coupon's expiration date

5. *percentage*: String representing the discount percentage

Any of these fields, except for *name*, can be $None$.
Next, we define the following weights:

$$
\begin{aligned}
w_n &= 0.4 \\
w_p &= 0.3 \\
w_{pr} &= 0.2 \\
w_v &= 0.1 \\
w_{po} &= 0.5 \\
w_{pn} &= 0.5
\end{aligned}
$$

We now define a function that assigns similarity scores to pairs of coupons:

$$
\begin{aligned}
S(c_1, c_2) = w_n R(c_1[\text{name}], c_2[\text{name}]) + \\
w_p w_{po} Q(c_1[\text{old\_price}], c_2[\text{old\_price}]) + \\
w_p w_{pn} Q(c_1[\text{new\_price}], c_2[\text{new\_price}]) + \\
w_{pr} R(c_1[\text{percentage}], c_2[\text{percentage}]) + \\
w_v R(c_1[\text{validity}], c_2[\text{validity}])
\end{aligned}
$$

where: - $R(s_1, s_2)$ is a similarity measure between two sequences, as defined in [**?**]. - $Q(x, y)$ is 1 if $x = y$ and 0 otherwise.

For each sample S_i, we associate a multiset of expected coupons C_i. Given a multiset $\tilde{C}_i$ obtained by the pipeline being benchmarked for an input sample, we apply the following procedure:

$lonely_i \leftarrow 0$
$similarities_i \leftarrow []$
for each $c \in \tilde{C}_i$ **do**
    $c' \leftarrow \arg\max_{c' \in C_i} S(c', c)$
    if $S(c', c) == 0$ **then**
        $lonely_i \leftarrow lonely_i + 1$
        continue

end if
$similarities_i.append(S(c, c'))$
$C_i.pop(c')$
end for
$lonely_i += len(C_i)$
for $i = 0$ **to** $lonely_i$ **do**
$similarities_i.append(0)$
end for
$similarity_i \leftarrow \text{mean}(similarities_i)$

Finally, we compute the average similarity across all samples.

Dataset

For benchmarking, we selected a representative subset of the data provided by Murmuras. This dataset is relatively small, containing fewer than 200 coupons, which enables fast evaluation. To improve the reliability of the benchmark, the data is sourced from multiple applications.

Proposed Solution

Input Data Format and Acquisition

Our solution will process input data in the form of CSV files representing the XML structure underlying visible screen content. These CSV files provide information about various aspects, including the location and content of text fields as well as their hierarchical structure.

During the fine-tuning process described below, we will use data provided by Murmuras. Since fine-tuning will be performed in a supervised manner, the expected model output will be generated with the assistance of ChatGPT. At runtime, the pipeline will process data extracted from smartphone screens using a scraping tool provided by Murmuras.

Currently, we are developing two different approaches to solving the problem.

Solution 1: Directly Querying an LLM

In this more straightforward approach, we will fine-tune the Llama3.2-1B model from the Llama family [**?**], selected due to its optimal size based on

the considerations outlined above, its liberal licensing terms, and its compatibility with fine-tuning frameworks [**?**]. The model will be executed using the llama-cpp tool [**?**][**?**], ensuring that its output is formatted in JSON. This structured output will significantly simplify postprocessing, reducing it to a trivial aggregation task.

Model Fine-Tuning

The fine-tuning process will be conducted in the Modal cloud environment [**?**]. To accelerate training, we will utilize the Unsloth framework [**?**], which is optimized for efficient fine-tuning of LLMs.

Solution 2: Labeling via Lightweight LLM

Our pipeline will consist of the following modules:

Preprocessing

We will segment the screen representation in CSV format into logical parts (e.g., individual coupons). This segmentation will be performed either using a clustering algorithm we developed or through fine-tuning a small LLM for this specific task.

Token Labeling

Using a fine-tuned model from the BERT family [**?**], we will assign classes to individual tokens that correspond to key coupon attributes (e.g., original price, discounted price, etc.).

Model Selection After preliminary research, we decided to focus on transformer models with parameter counts ranging from 10 million to approximately 500 million. Most of the models we selected are derivatives of BERT [**?**]. The three main subtypes under consideration are:

1. BERT: ~100 million parameters
2. DistilBERT: ~65 million parameters
3. ALBERT: ~11 million parameters

The primary advantage of these models is their relatively small size. Previous experiments [**?**] have demonstrated that the LLaMA-2 model, with 7 billion parameters, can be deployed on mobile devices. Models from the BERT family are an order of magnitude smaller, making them more suitable for our use case.

Postprocessing

Postprocessing will involve grouping the labeled data into structured coupon representations.

Deployment on Mobile Devices (Optional)

We plan to develop a standalone mobile application or integrate the functionality into an existing one, implementing the aforementioned processing pipeline. A background service will be designed to process incoming data in real time. However, if real-time processing proves to be too resource-intensive, an alternative approach will be adopted in which the application stores the data temporarily (with an estimated daily storage requirement of approximately 15KB, assuming 15 minutes of daily coupon application usage) and processes it during off-peak hours, such as at night when the device is idle.

The application will be designed to ensure compatibility with Android 9 and later versions. However, achieving universal compatibility across all devices is not a strict requirement. Data storage and further transmission will be managed by the company's existing application infrastructure at this stage.

Milestones

Research

Planned Completion: 30.11
By the end of November, we aim to have selected the architecture, a specific model, and proposed algorithms for preprocessing and postprocessing.

Proof of Concept

Planned Completion: 31.12
We plan to create a prototype application demonstrating the full functionality.

Idea Gathering for Improvements

Planned Completion: 31.01
January will be a month during which we do not plan intensive work on the project due to exams. We will use this time to potentially finish previous milestones and reflect on the project's direction.

Solution Finalization and Testing

Planned Completion: 30.04
At this stage, we will focus on improving the solution, fixing bugs, and testing.

Bachelor's Thesis

Planned Completion: 30.06
We will focus on writing and refining the bachelor's thesis.