UNIVERSITÀ DEGLI STUDI DI UDINE
HIC SUNT FUTURA

DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS

MASTER THESIS IN
INFORMATICA

Fine-tuning Large Language Models for Gamified Urban Mobility Recommendations

CANDIDATE
Belliato Riccardo

SUPERVISOR
Prof. Kevin Roitero

CO-SUPERVISOR
Dott. Antonio Bucchiarone

TUTORS
Dott.sa Annapaola Marconi
Dott. Federico Bonetti

Academic Year 2022-2023

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Alla mia famiglia
Alla mia comunità
Ai miei amici
Uno speciale rigraziamento a Nicola, Silvia, Marianna, Emma, Gioele, Emanuele e Lia per l'ospitalità durante le trasferte.

Abstract

In response to the growing global emphasis on sustainable living and smart cities, multiple initiatives leveraging cutting-edge technologies like Artificial Intelligence and Social Computing have been introduced by governments and corporations. Within this framework, the MoDiS Research Unit at the Fondazione Bruno Kessler in Trento has developed 'Play&Go', an innovative and automated system designed to set sustainable mobility goals for users.

This thesis presents the development and implementation of a recommendation model powered by a custom adaptation of a pre-trained Large Language Model (LLM). We leverage the data from the 'Play&Go' database into textual prompts to perform an ad-hoc finetuning of the LLM, by setting the model's objective to predict the likelihood of users accomplishing a set of sustainability goals based on their history of achieving previous objectives. Our approach has been evaluated against other state-of-the-art models in various scenarios. Results show that LLMs are effective in performing effective recommendations in a gamified urban mobility setting.

Contents

List of Tables

List of Figures

I

State of the Art and Context

1

Introduction

The purpose of this chapter is to briefly introduce the work carried out in this thesis. In particular, Section 1.1 describes the context in which it is situated. Subsequently, Section 1.2 outlines the overall objectives to be achieved. Finally, Section 1.3 provides a synopsis of the following chapters.

1.1 Context

This thesis is collocated in the context of *Recommender Systems* (RS) and *Deep Learning* (DL). A definition of Recommender System has been written by Ricci et al. [2010]:

> Recommender Systems, called also Recommendation Systems, are software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read.
>
> Development of recommender systems is a multi-disciplinary effort which involves experts from various fields such as Artificial intelligence, Human Computer Interaction, Information Technology, Data Mining, Statistics, Adaptive User Interfaces, Decision Support Systems, Marketing, or Consumer Behavior

Real-world examples of recommender systems are platforms like Netflix, Spotify, YouTube or Amazon. In the case of Netflix the platform employs a sophisticated RS to analyze user viewing history, ratings, and viewing habits. By leveraging these data, the RS can suggest movies or TV shows that align with the user's taste, introducing the users to new content, and enhancing their overall viewing experience. This not only keeps users engaged but also contributes to increased user satisfaction and platform retention [Gomez-Uribe and Hunt, 2015].

The literature presents different types of RS, each with its strengths and limitations; the choice of one type over others (or a combination of several of them) depends on various factors such as domain characteristics, data modeling and availability, performance constraints, *etc.*. The main types of RecSys are *Collaborative Filtering* and *Content Based filtering* [Ricci et al., 2010].

Collaborative Filtering is a type of RS that makes predictions about user's interests by collecting preferences from many users (user-item interactions). It Identifies patterns and similarities between

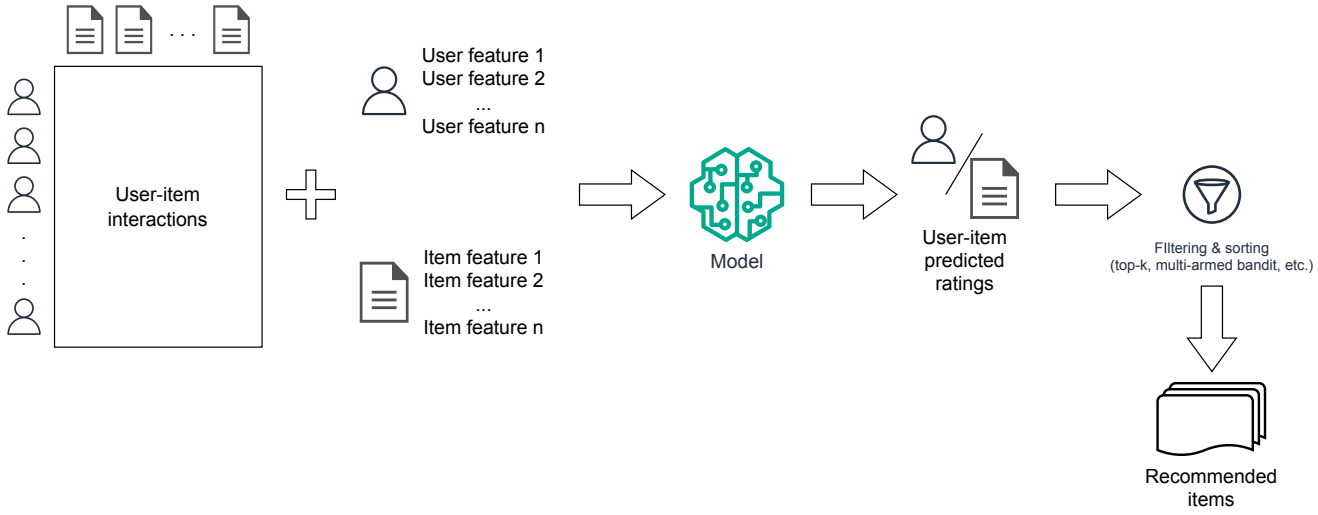

Figure 1.1: General architecture of RecSys.

users or items to recommend items that users with similar preferences have liked or interacted with [Ricci et al., 2010, Chapters 4 and 5].

Content-Based Filtering recommends items based on the characteristics of the items and a profile of the user's preferences. It involves analyzing the content of the items and creating a user profile to suggest items that match the user's preferences [Ricci et al., 2010, Chapter 3].

The diagram in Figure 1.1 illustrates the general workflow of a RS. The system takes into account information about user-item interactions (such as clicks, explicit ratings, showtime, etc.) and information about the users and/or items themselves. This information is then used to feed a model that predicts a rating for each user and item. Real and estimated ratings are used in order to recommend items. One method is top-k, which selects k items with the best ratings. There is another family that is the bandit-based methods, which combine top-k with randomness to suggest items that may not have been initially considered but could be liked by the user [Ricci et al., 2010].

This thesis focuses on ratings prediction models, specifically Sequence-aware, a type of Content-Based Filtering method which, based on the user's past interactions, predicts the next action of the user [Quadrana et al., 2018].

The models used to predict evaluations are essentially machine learning models, with the most powerful being those based on deep learning methods. Machine learning is a branch of artificial intelligence (AI) that uses statistical methods to enable machines to improve with experience without explicit programming. Classical machine learning methods include logistic and linear regression, support vector machines, and decision trees. According to LeCun et al. [2015] conventional machine learning techniques were limited in their ability to process natural data in their raw form. In fact, they require to preprocess raw data in a suitable internal representation from which the model could handle the required task, so in recent years representation learning methods have been developed.

> Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification.
>
> Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the

> representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. [LeCun et al., 2015]

The main deep learning methods are Multilayer Perceptrons (MLP), the simplest and most general method, Convolutional Neural Networks (CNN), used for processing images and 2D data, and Recurrent Neural Networks (RNN), suitable for sequential data such as text or audio [Zhang et al., 2023a]. Currently, the state of the art in this field is represented by Transformers [Vaswani et al., 2017], which are an improvement of RNNs, and Graph Neural Networks (GNN), which are able to handle data in graph form [Wu et al., 2020].

A ML/DL model can be thought of as a parameterized generic code in which the values of the parameters are obtained by optimizing the value of a function called *Loss Function*, which measures the magnitude of the error between the output of the model and the expected result. The loss function must be appropriate to the problem we are trying to solve. A model is as complex as the number of its trainable parameters. In the case of the most advanced DL models, such as language models, millions or even billions of parameters can be reached.

The process of calculating the value of the parameters is called *training*. Examples of Loss Functions are Mean Squared Error, typically used in regression tasks such as price forecasting, and Cross-Entropy, suitable for classification tasks such as image recognition [Zhang et al., 2023a]. The training process is done using an algorithm called *Gradient Descent* that adjust parameters' values iteratively using the partial derivatives of the cost function with respect to each parameter until it converges or after a maximum number of iteration. The partial derivative is used because it indicates in which direction the cost function decreases more rapidly [Zhang et al., 2023a].

A trained model must be validated by measuring its performance on data that are different from the data used for training. Performance is measured by defining a set of suitable metrics. There are two main methods for validation. One of them involves splitting the training data into two datasets: the training set, used in the training phase, and a test set, used for validation. This method is the simplest, but is not recommended in the case of limited data and it is sensitive to bias in the test set [Zhang et al., 2023a]. To address these issues, cross-validation is used. In cross-validation, the training data is split into three or more subsets, typically five or ten, called folds. For each fold, a new train and test phase is launched to aggregate and compare the metrics obtained. The way folds are created and used depends on the cross-validation variant used [Zhang et al., 2023a]. In this thesis, we employed two different types of cross-validation, which are described in Chapter 3. The main advantage of cross-validation is that it offers a more precise estimation of the model's performance on new data, which is crucial for evaluating the model's generalizability, and it helps to prevent overfitting, which is a common issue in machine learning. Overfitting is a problem whereby the model fits the training data too closely. This can lead to good performance on the training data but poor performance on the test data. It occurs when the chosen model is too complex for the desired task or when the quality of the training data is poor [Zhang et al., 2023a].

1.2 Goals

This thesis presents an alternative approach to recommendations in the context of Sustainable Urban Mobility. In particular, we aim to involve users to embrace more environmentally sustainable behaviors. One way to do this is to challenge users by suggesting some goals and rewarding them for achieving them. This is a common method in the *gamification* domain and it will be discussed in more detail in Section 2.1.

Therefore, our aim is to suggest objectives to users based on their mobility habits, encouraging them to achieve these without finding them too challenging or too simple, and promoting long-term behavioral changes.

To achieve this, **we use a cutting-edge Artificial Intelligence and Natural Language Processing model to predict and evaluate a user's likelihood of completing a given target from a set of candidates, based on their previous outcomes, in order to filter and recommend the most promising ones**.

To validate our work, we trained and tested our model on a dataset of real data. We compared our model with similar ones using a cross-validation approach and various metrics.

1.3 Synopsis

Chapter 2 will introduce the state of the art of the topics covered in this thesis. In Section 2.1 we will introduce the concept of *gamification*, why it is important and the work done by the MoDiS Research Unit in this domain. In Section 2.2 we will describe and analyze Play&Go, a platform developed by MoDiS for implementing sustainable mobility campaigns based on gamification methods. In particular, Subsection 2.2.3 will be dedicated to describe the recommendation algorithm currently implemented in Play&Go. Then, in Section 2.3 we will describe the main Deep Learning models used to process sequential data and how they are used to build RS. In Section 2.4 we will give a brief presentation of some research which uses machine learning to make predictions about user mobility. In Section 2.5 we introduce Large Language Models and their use in the context of RS with examples taken from the literature.

Chapter 3 will introduce and describe tools and methods used in our experimental setup. Section 3.1 introduces the model and the evaluation methods followed during the experiment. Details on how the model was trained and how it generates the output are specified in Section 3.5. In Section 3.2 we describe the baseline, in particular the Deep Learning models we used to compare the performance of our model. Sections 3.3 and 3.4 will provide a detailed and comprehensive description of the data set used in our experiment, including an exploratory analysis, how it was constructed from Play&Go data and converted to text prompts. The chapter concludes with Section 3.6, in which we describe the metrics used to evaluate the performance of the models subject to our experiments.

In chapter 4, guided by some research questions introduced in Section 4.1, we will analyze the results obtained. In Sections 4.2, 4.3, 4.4 we will answer the research questions comparing the metrics obtained by the model with the metrics obtained by training the models in the baseline with the same data, while in Section 4.5 we will analyze the predictions made by the model during the experiments to find possible biases or weaknesses.

Finally, the chapter 5 will contain a recap of the experimental results and a discussion about limitations and possible future developments of the work presented in this thesis, in particular how the model can be integrated into Play&Go and how the raw model output can be used for other domain tasks, e.g. difficulty prediction, in-game score calculation, top-k extraction, *etc.*.

2

Related Works and Background

In this chapter, we will analyze the context in which this thesis is inserted. Firstly we will describe the technique of *gamification* and why it is important today.

Then we will analyze Play&Go, a platform for implementing sustainable mobility campaigns based on gamification methods, and how it was integrated into the Air-Break project promoted by the City of Ferrara. A particular focus will be reserved on the backend of the platform, where a full automatic framework generates and recommends sustainable mobility challenges for users, and how we will try to improve it with our work.

Finally, we will present some works from the literature that have used deep learning models to predict user intent in mobility and the main AI tool we used for our work, the Large Language Models, and how they can be used to build RSs, presenting some literature examples.

2.1 Gamification

According to the International Energy Agency (IEA), in 2018 transport accounts for around one-fifth of global carbon dioxide (CO_2) emissions and road travel accounts for three-quarters of transport emissions. Most of this comes from passenger vehicles – cars and buses – which contribute 45.1%. The other 29.4% comes from trucks carrying freight [Ritchie, 2020]. In recent years governments and local authorities have promoted initiatives to promote more enviromentally sustainable behaviors. One of these initiatives is the Air Break project, promoted by the city of Ferrara.

In this context, one of the most promising approaches are *gamification*-based techniques. Gamification can be defined as *"the use of typical elements of games in a non-game context"* [Caponetto et al., 2014, Bassanelli et al., 2024]. Game elements are the basic components of games. They include in-game points, rules, modality of cooperation or competition, goals, leaderboards, etc. and they can be classified according to various taxonomies [Bassanelli et al., 2024]. In Figure 2.1 the taxonomy proposed by Toda et al. is represented.

When designing gameful systems, various aspects has to be taken into account. The first aspect to consider is whom the gamification is directed and what the characteristics of the target group are. It is important to consider personal and demographic differences, and select the right game elements when the target reflects a more specific population [Bassanelli et al., 2024]. One of the most famous theory

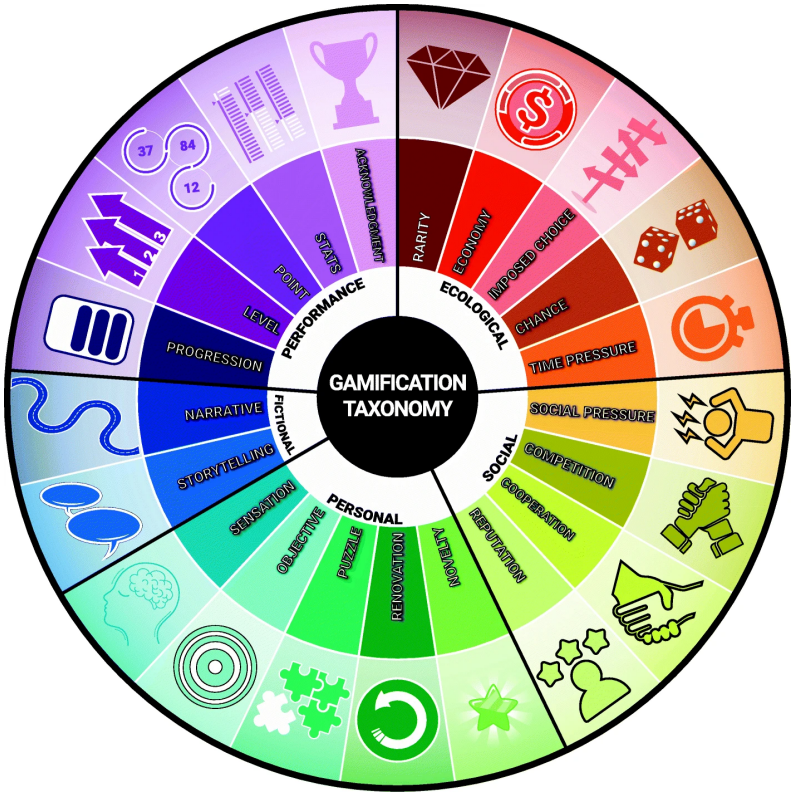

Figure 2.1: Game design taxonomy (taken from Toda et al. [2019]).

in gamification are the Self Determination Theory, which states that people are intrinsically motivated leveraging the satisfaction of three basic needs: autonomy, competence and relatedness. How these aspects were leveraged depends on individuals [Ryan and Deci, 2000], so people can be motivated and keep engaged by different aspects of the same activity, and this should be taken into account when designing gamified applications.

The second aspect is the mechanism by which goals can affect users' performance. Bassanelli et al. cite four main mechanisms: (i) the goals aim to focus the player only of relevant actions at the expense of non-relevant actions, e.g. "use object A instead of object B"; (ii) the goals in which the player is invited to increment his performance with hard goals; (iii) the goals designed to affect the persistence of the players and (iv) the goals which aim the player to discover elements and strategies.

The third aspect is the game modality, which is the structure of user interaction and the choice of objectives. Understanding the impact of modality on the effectiveness of gamification is crucial, as it helps designers select the most suitable type of interaction between users. There are four main modalities: individual, cooperative, competitive and cooperative-competitive [Bassanelli et al., 2024].

Another aspects to take into account are the domain and the context in which the system has to be inserted, the technology to use and a suitable feedback mechanism. It is crucial to provide information to users taking into account their preferences to optimize the effectiveness of gameful systems [Bassanelli et al., 2024].

In a gamified application, one of the main concept is the *challenge* (or *quest*), a "unit of playable content that sets a demanding goal that a player should achieve – under temporal or other constraints – in exchange for a prize or reward in the game" [Khoshkangini et al., 2021]. According to Khoshkangini et al. [2021], well-designed challenges have been empirically proven to effectively maintain user engagement.

However, manually designing and assigning challenges to users requires significant effort. Therefore, automatic approaches must be exploited. The main limitation of automatic challenge assignment is the lack of diversity and personalization, which can lead users to lose interest over time.

Countering these issues and developing gamified application for various domains are two of the main topics researched in Motivational Digital Systems Research Unit (MODIS) led by Dott.sa Annapaola Marconi and based in Fondazione Bruno Kessler (FBK) in Trento, Italy. One of the most successful and long-lived application developed by MODIS is Play&Go, a gamified platform for Sustainable Mobility[1].

2.2 Play&Go

2.2.1 The AirBreak project

Air Break[2] is a project funded under the European Urban Innovative Actions[3] programme, which has seen Deda Next[4], the University of Ferrara, the Politecnico di Milano, SIPRO (Provincial Agency for the Development of Ferrara)[5], Hera Group[6], Lab Service Analytica[7] and Fondazione Bruno Kessler at the forefront alongside Local Administration [FBK Editorial Staff, 2023].

The partners, led by the Administration, were asked to test innovative solutions that would contribute to the reduction of 25% air pollution in selected areas of the city and to carry out educational activities to involve and raise awareness in all citizens of issues such as commute, lack of green spaces, and lack of up-to-date information on air quality[8], encouraging them to make sustainable daily choices [FBK Editorial Staff, 2023].

The air quality data were collected using an IoT infrastructure built by Deda Next and Lab Service Analytica. Through the infrastructure, which extends the municipal Local Information System and will continue operating after the project is completed, the municipality now has access to 11 million new data on the air quality and travel habits of its citizens, which can also guide it in the development of future public policies [FBK Editorial Staff, 2023]. These data have been made available to the citizen and can be consulted from a dedicated dashboard[9] developed by Deda Next and Politecnico di Milano.

One of the core objectives of the Air Break project is informing and heightening citizens' awareness about sustainable mobility services, with the ultimate goal of *fostering the adoption of eco-friendly travel habits*. In outlining the project's scope, various campaigns have been designed to address four key objectives.

AirBreak aims to **actively engage citizens**, raising awareness of existing and new mobility resources, and highlight the impact of daily mobility choices. The goal is to establish a new cultural norm for urban and rural mobility.

[1] Sustainable mobility is defined by Commission [1992] as "a strategy to integrate transport into an overall pattern of sustainable development which would seek to meet the needs of the present without compromising the ability of future generations to meet their own needs"

[2] https://airbreakferrara.net/

[3] https://www.uia-initiative.eu/en

[4] https://www.dedanext.it/

[5] https://www.siproferrara.com/

[6] https://www.gruppohera.it/

[7] https://www.labservice.it/

[8] https://airbreakferrara.net/wp-content/uploads/2021/03/comunicato-stampa-comune-ferrara_uia-air-break.pdf

[9] https://airbreakferrara.net/che-aria-tira/

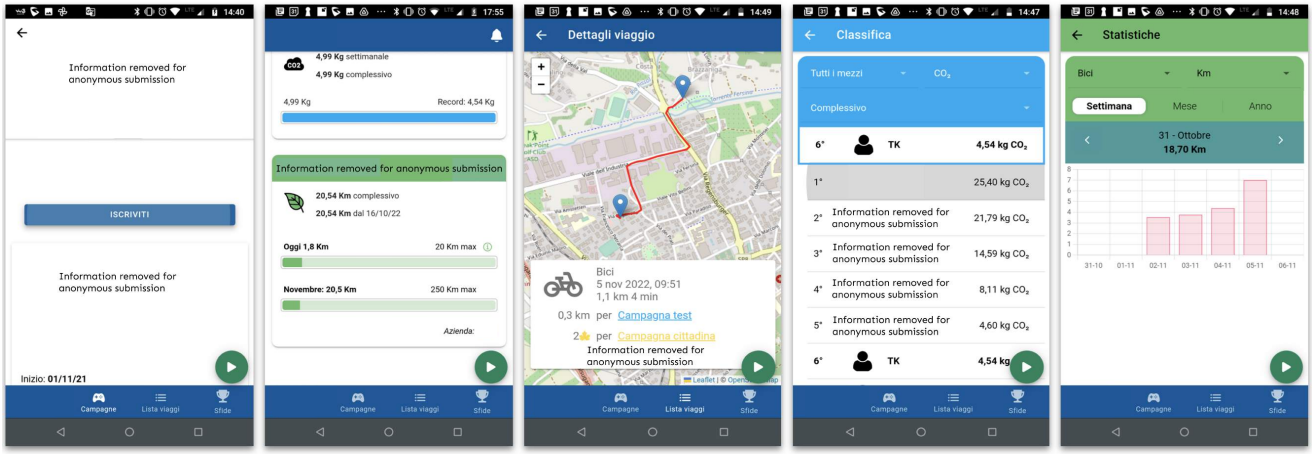

Figure 2.2: The Play&Go application UI.

AirBreak seeks **voluntary travel behavior change** through a mix of virtual (game-based) and tangible incentives, utilizing personalized cooperative and competitive game mechanics. The focus is on encouraging and sustaining the adoption of sustainable mobility habits.

Additionally, AirBreak prioritizes **community building** by actively involving citizens in initiatives and events. The project aims to create a local community of users through customized behavioral change programmes targeting various segments, including students and employees. This fosters collaboration and support among community members.

Lastly, AirBreak emphasizes producing **measurable outcomes** by assessing the impact in terms of users' engagement, retention, awareness, and behavioral change. Insights from these outcomes will inform refinements in subsequent project phases, ensuring continuous improvement based on real-world experiences.

Within the AirBreak project framework, various sustainable mobility campaigns have been designed and executed, targeting specific scenarios like *home-to-work*, *home-to-school*, leisure, and *free time mobility*. These initiatives cater to various user groups, including the general public, students, and employees.

AirBreak mobility campaigns are facilitated through the Play&Go[10] platform. This digital platform supports the definition and management of sustainable mobility campaigns, allowing customization of mobility objectives and target users. Customisation options include the means of promotion of mobility, travel validation criteria, implementation of competitive/cooperative game elements, and real incentives provided by the campaign [Kazhamiakin et al., 2021b].

2.2.2 The Platform

Play&Go is a a full-fledged Gamification Platform supporting the design and execution of a long-running gamified urban mobility campaign utilizing gamification to encourage voluntary changes in travel behavior. Users engage with the platform through the Play&Go App (see Figure 2.2), which offers various functionalities such as player registration, discovery and registration for active campaigns in their area, profile management, tracking of sustainable trips, viewing campaign results, accessing rewards information and reviewing rules and regulations of different campaigns [Bonetti et al., 2023].

[10] `https://playngo.it/`

A **Urban Mobility Campaign (UMC)**, is an initiative that encourages citizens to actively adopt sustainable transportation through gamification. To participate to campaigns, individuals must register and use the Play&Go app to track eco-friendly trips, which are essential components of engagement. Three UMCs were implemented within the AirBreak project[FBK Editorial Staff, 2023, Bonetti et al., 2023].

Bike to Work [Bucchiarone et al., 2023] is a sustainable mobility initiative that targets public or private sector employees who commute from home to work and aims to promote the use of bicycles for home-work trips by providing economic incentives to employees.

High School Challenge engages high school students in a competitive class-level challenge. The performances of each student in the main game will contribute to the results of the class in the high-school competition. The class with the best results, will be awarded with a class-level prize.

The **Kids Go Green** program uses game-based education to involve children of primary and middle school age with their teachers and families. In Kids Go Green, the sustainable kilometers made by each child on their trips from home to school contribute to the progress of the whole group of children on a virtual educational journey to places in the real world that is personalized according to the group interests and capabilities.

At the heart of the UMC gaming experience are the *Eco-Leaves Points*, also called *Green Leaves*. These points play an important role in progressing on both weekly and global *leaderboards*, influencing the distribution of weekly and final prizes. Earned for each *tracked and validated sustainable trip*, the calculation depends on factors such as the distance covered and the sustainability of the chosen means of transportation.

The fundamental mechanics of the game center on trip tracking, where players specify their mode of transportation, including walking, biking, public transportation, or carpooling. An automatic validation process ensures the accuracy of each trip, preventing potential game abuse Kazhamiakin et al. [2021b].

The game seamlessly incorporates both *single-player* and *multiplayer challenges*, catering to a diverse range of player preferences. Single-player challenges encompass various types, including *performance-based* - e.g. "Walk at least 10 Km to obtain 150 Green Leaves" -, *repetitive behavior* - e.g. "Collect 50 Green Leaves for 5 days in a row to obtain 300 Green Leaves" - , *surveys*, and *events*, all contributing to the player's accumulation of Green Leaves points.

In multiplayer challenges, players enjoy a sense of community and connection. In group challenges, the goal is the same for the two players, while the reward may differ due to the specific difficulty of the challenge for each player. These challenges come in cooperative, time-based competitive, and performance-based competitive modes [Kazhamiakin et al., 2021a].

Cooperative challenges require collaborative efforts from participants, fostering a spirit of teamwork. In the cooperative mode, the two players' performances were added in order to reach the goal before the challenge expires. The reward is the same for both. An example of a cooperative challenge is: "Walk at least 10 km between you and Player2. If you win, you will both get 150 Green Leaves". Meanwhile, *competitive challenges* add an engaging dimension with fixed targets, time constraints, and performance-based criteria. In time-based competitive challenges, the target is fixed, and the winner is the player that is fastest in reaching it. In this case, the reward is calculated based on individual players, so it can be different between them. An example of such a challenge is: "Walk 10 km before

your opponent Player2. If you win, you'll get 100 Green Leaves. If Player2 wins, she-he will get 150 Green Leaves". In Performance-based competitive challenges, instead, there is not a specific goal: upon expiration, the player that has the highest performance wins and the reward is a booster calculated on the weekly points obtained - e.g., "Walk more km than your opponent Player2. The winner will get a 30% bonus of all the points gained during the week".

Challenges' goals can be either in-game performances or specific behaviors promoted by the game, such as breaking one's habits, e.g., in the case of the UMC trying a new way of commuting, or increasing one's usage of sustainable transportation means.

An integral element of the UMC involves the ongoing assessment of player performance through *game levels*. Players advance through distinct levels, including *Green Lover*, *Green Warrior*, and *Green Guru*, determined by the cumulative points they accumulate. In particular, this system ensures that specific game features become accessible to players once they reach designated levels, introducing a sense of progression and achievement to the gaming experience.

The gameplay follows a *weekly* structure, fostering a sense of regularity and enables the creation of *weekly leaderboards*. These leaderboards, which reflect performance within a given timeframe, contribute to the excitement of weekly prizes sponsored by local partners. Crucially, this format ensures that newcomers have a fair opportunity to compete with experienced players, leveling the playing field, and maintaining a vibrant gaming community.

To encourage continuous engagement, the UMC concludes with final prizes bestowed upon top players on the global leaderboard. Both weekly and final prizes are generously provided by local associations and sponsors, enhancing the overall allure of the campaign.

The platform also includes a set of end-user applications supporting the management of the whole game life–cycle: from game definition and deployment, to the supervision of the campaigns, and the analysis of game impact, both in terms of end-users participation and in terms of sustainable mobility. For doing this the platform exposes an API, from which some Web Apps were developed [Kazhamiakin et al., 2021a].

The Game Definition Web App supports the game designer in the definition of the game logic and its deployment. In particular, the game designer can exploit this application to define the player actions supported by the game, the game concepts and mechanics, that is, points, badge collections, challenge models, levels, leaderboards, and dynamics, i.e., game rules.

The Game Supervision Web App permits to inspect the players' performances in terms of valid trips and km tracked in the various mobility modes, but also the Green Leaves earned, the game levels reached, as well the won, lost and on-going challenges.

The Itinerary Validation Console allows the game administrator to inspect all tracked trips. This tool provides a map-based visualization of the tracked itinerary in order to detect the results produced by the validation algorithm, in order to inspect and solve specific issues raised by players, e.g., validity outcome for a certain trip or complaints about the number of points awarded for a specific trip. The console is also useful to promptly detect cheaters and take actions, from sending warning messages to banning them from the game.

Lastly, the platform also provides an open source web-based dashboard that ingests, processes, and provides visual representations (maps and infographics) of game-related and mobility-related analytics,

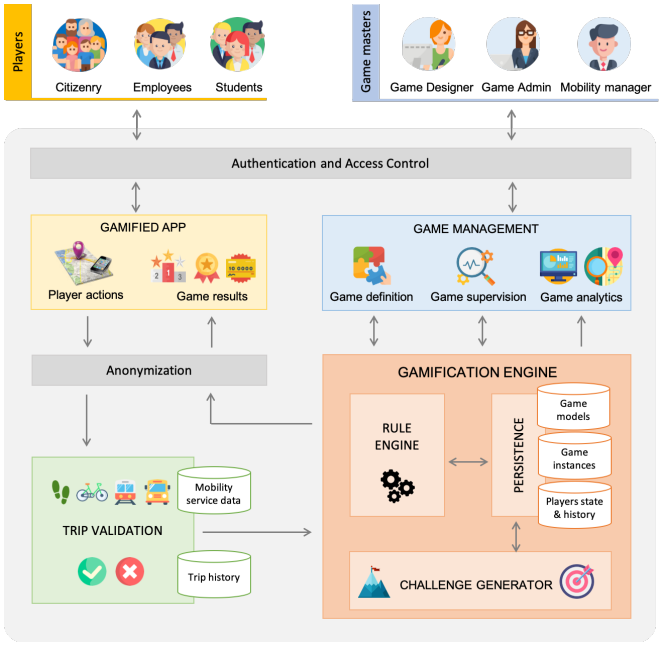

Figure 2.3: Conceptual schema of Play&Go Platform (taken from `https://modis.fbk.eu/playandgo/`)

based on heterogeneous raw data, such as game logs, tracked trips, and existing mobility-related data sources [Kazhamiakin et al., 2021a].

A summary of the structure and the functioning of Play&Go platform is provided in Figure 2.3.

2.2.3 The Gamification Engine

The "core" of Play&Go is the *Gamification Engine*, which periodically assigns new challenges to users. As reported in Section 2.1, design and assign manually challenge is an unfeasible task, therefore a possible solution is to exploit automatic techniques. One of these is *Procedural Content Generator* (PCG).

Procedural Content Generation is "a set of techniques designed to automate the construction of Game Design Elements (GDEs), such as game items, encounters, buildings, or even whole levels" [Scanagatta and Marconi, 2021]. PCG has already been applied successfully in computer games [Sanner et al., 2023]. In our case we aim to use PCG to generate personalized mobility challenges for our user base to enhance long-term participation and induce positive change in users' behaviors. In this chapter, and in general in this thesis, we will focus on PCG for single-player challenges.

Play&Go implements an automatic PCG (APCG) framework developed by Khoshkangini et al. [2017, 2021]. It is defined as an "online, constructive, and parametrized case of PCG applied to gamification that results in the automatic generation and injection of personalized and contextualized units of playable content." [Khoshkangini et al., 2021].

It is composed of two main parts: the PCG itself, which generates a pool of candidate challenges for each user, estimates their difficulty, and calculates the in-game reward for each one based on the user's performance, both individual and global, and an RS, which filters and sorts the candidates to recommend the best ones.

The PCG part contains three subparts: the *Challenge generator*, the *Difficult estimator* and the *Reward calculator*, while the RS part is composed by a Machine Learning module, which estimates the probability of success for the challenges, and a *Filtering&Sorting* (F&S) module, which applies some

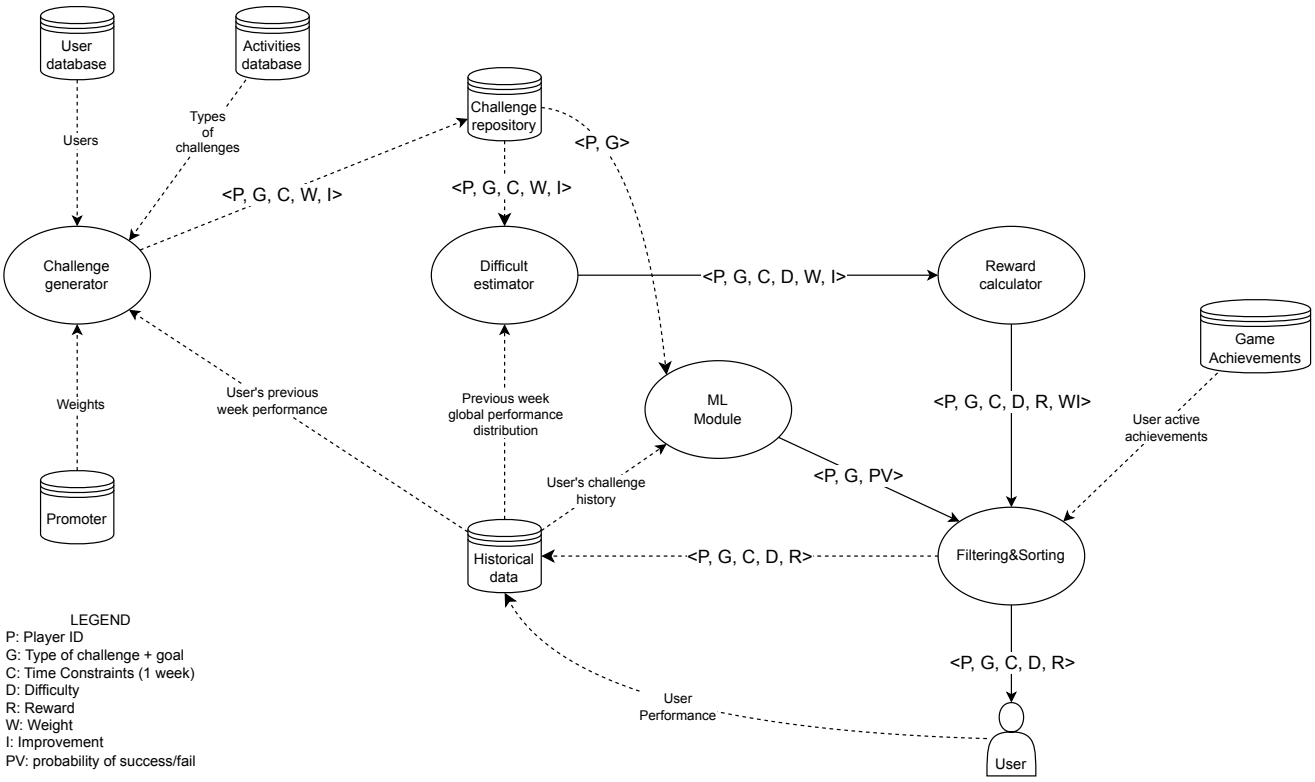

Figure 2.4: The APCG workflow. The ellipses represent the modules, the cylinders represent data storages. The data flow between components is visible in the arrows.

heuristics to sort the challenges and return the recommendations. All of these components will be described in next paragraphs.

In APCG the challenge are represented as tuples $\langle P, G, C, D, R, W \rangle$ where P is the player, G is the challenge, C the constraints - e.g. a temporal deadline -, D the difficulty, R the reward, W the *Weight*, a numerical value which capture how the goal G is important for the promoter of the campaign. The parameter G has two sub-parameters: the mobility indicator MI - the "type" of challenge - and T, the target value to be reached to complete the challenge.

Other important values used in the process are I, the relative improvement of the challenge with respect to the actual performance level, the *Weighted improvement* WI, the product of W and I, and the probability of success of the challenge PV.

The workflow of the APCG framework is summarized in Figure 2.4.

Challenge Generator

The purpose of this component is to generate a set of potential challenges to suggest to each player.

The Challenge generator takes as input three sets: the players, the mobility indicators and the weights, which associate for each mobility indicator their weight. For each player, we know his challenge history and his performance during the previous weeks for each mobility indicator.

One of the most important aspects in PCG is *difficulty calibration*. The difficulty of proposed goals should be calibrated to the player's performance, as it should be proportional to their skill increase. In jargon, we said that we want to reach a state of *flow*, "where the player is fully engaged with the system - that is, when the challenges match the skill level of the player. Underestimation creates boredom,

Algorithm 1 Challenge generator pseudocode.

Require: Ws Weights set
  $S \leftarrow \{\}$ ▷ Set of generated challenges
  for all $P \in \{P1, P2, \ldots, P_n\}$ **do** ▷ Players
    for all $MI \in \{\texttt{Walk\_Km}, \texttt{Train\_Km}, \ldots\}$ **do** ▷ Mobility indicator
      $C \leftarrow$ 'one week'
      $W \leftarrow Ws[MI]$ ▷ Weight for MI
      if P has done any activity for MI during previous week **then**
        $T_{old} \leftarrow P[MI]$ ▷ Last week performance for the MI indicator of player P
        for $I \leftarrow 10\%, 20\%, \ldots, 100\%$ **do**
          $T \leftarrow T_{old} + I$ ▷ Percentage increment
          $G \leftarrow \langle MI, T \rangle$
          $S \leftarrow S \cup \langle P, G, C, W \rangle$
        end for
      else
        $T \leftarrow 1$
        $G \leftarrow \langle MI, T \rangle$
        $S \leftarrow S \cup \langle P, G, C, W \rangle$
      end if
    end for
  end for
  return S ▷ To Challenge repository

while overestimation leads to anxiety or frustration" [Scanagatta and Marconi, 2021].

For each player P and for each mobility indicator MI if P has done any activity for MI - for example, the player walked for 1 Km, but he failed his challenge - the performance is used as baseline and ten different challenges with growing improvement - from 10% to 100% - are instantiated. For the other mobility indicators the algorithm instantiates one challenge with 1 as target value.

All challenges have as constraint C one week and they include the weight W of the mobility indicator and the improvement factor I. Then all challenges are saved in a database called *Challenge repository*, ready for the Difficult estimator. The process of generating challenges is summarized in Algorithm 1

A variant of the algorithm is described by Scanagatta and Marconi [2021], which was used during the 2019-2020 edition of the game.

In this variant, a challenge is generated for each mobility indicator with a target calculated as $T = E * I$, where $I = 1.30$ and E is an estimation of the player's performance calculated using a Weighted Moving Average of their previous 5 weeks' performances, instead of just the previous week.

The choice of the value of I and the method of calculation of E is the result of a study in which Scanagatta and Marconi asked users to predict the performance of some players given their previous six weeks performances and minimizing the error between the proposed method and the mean of the responses.

Difficult estimator and reward calculator

In APCG difficult estimation consists in assigning a label among 'Easy', 'Medium', 'Hard' and 'Very hard' applying this method: given a challenge mobility indicator and a candidate player, the module takes into account the distribution of all player's past performance related to the considered indicator.

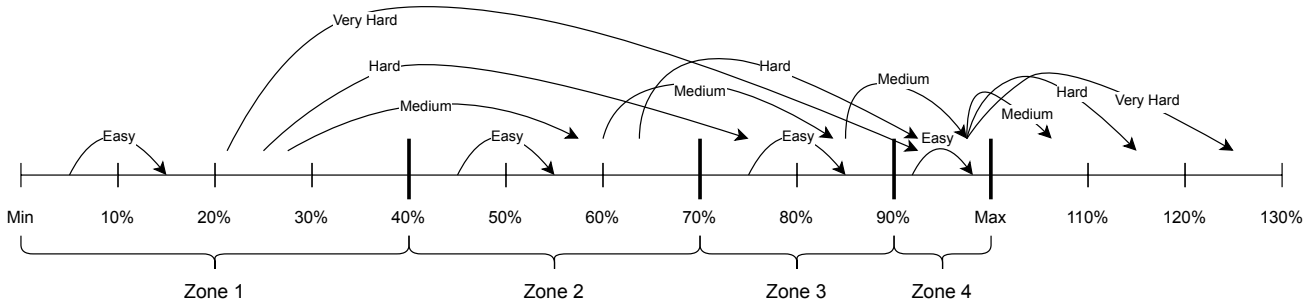

Figure 2.5: Difficulty estimator method.

The distribution is divided into four zones based on its deciles[11]. The first zone includes values from the global minimum to the 4th decile, the second zone includes values from the 4th to the 7th decile, the third zone includes values from the 7th to the 9th deciles, and the fourth zone includes values from the 9th decile to the global maximum.

The difficulty level D is determined by comparing the player's performance zone with the zone of the performance required by the challenge objective following the schema in Figure 2.5. The level is classified as 'Easy' if both are in the same zone, 'Medium' if the required performance is one zone away from the actual performance, 'Hard' if it is two zones away, and 'Very Hard' if it is three zones away. If a challenge request exceeds the global maximum, subsequent zones are set by increasing the maximum with the range of values between the 9th and the 10th deciles. For instance, if this range goes from 10 to 12 (i.e. decile range 3) and the challenge necessitates moving from 11 to 15, it will be classified as 'Hard' since it involves moving two zones higher of size 3: 10 to 12, 12 to 14, and 14 to 16.

Finally, the reward value R is assigned based on a table that considers the difficulty level D and improvement ratio I. The rewards range goes from 100 for an Easy challenge with 10% improvement to 250 for a Very Hard challenge with 100% improvement.

ML module and F&S

These modules compose the RS part of the framework. They take as input the labeled challenges and return a subset of these that will be assigned to the players.

The initial module is the Machine Learning module, which implements a Naive Bayes binary classification model trained on players' past challenges. The task is to estimate the probability PV that the player will complete or fail a candidate challenge, knowing the player's challenge history from the training data.

The second module is the 'Filtering & Sort' module, which implements a multi-criteria sorting algorithm that leverages both the user's and promoter's interests. The aim is to achieve the promoter's goals, such as increasing bike usage in the population, while also considering the user's goals, such as a user who does not have a personal bicycle, to avoid neglecting them. Therefore, it is important to find a balance between the two.

To achieve this, the algorithm operates as follows: each player's candidate pool is divided into two subsets. The first subset contains challenges whose rewards are sufficient for the player to reach any

[11] A decile is a statistical measure used to divide a set of data into ten equal parts, or groups, each representing 10% of the total. To calculate deciles, you arrange the data in ascending order and then split it into ten equal parts. The values that mark the boundaries of these parts are the deciles.

game achievement, while the second subset includes the remaining challenges. The two subsets are then sorted. The first subset is ordered by difficulty in ascending order, followed by descending order of Weighted Improvement and PV value. The second subset is ordered by difficulty, reward, Weighted Improvement and PV value. The final sorting is achieved by inserting the challenge from the first subset at the beginning of the list and placing the remaining challenges in the tail.

Limitations and discussion

Khoshkangini et al. outlined that the framework is designed to be easily generalized to other gamification domains, by defining new types of challenges instead of the mobility indicators, and during the A/B testing it showed that the challenges assigned by the automatic framework may be conducive to higher level of improvement for the same unit of reward compared to analogous challenges assigned manually through expert judgment, but some limitations are also highlighted.

Ignoring considerations about the A/B test itself, which are beyond the scope of this thesis, the authors report a lack of personalisation and diversity in both challenge assignment and proposed rewards. In fact, the framework does not consider players' characteristics or their physiological signals as features, such as the time in which the player completes challenges or his improvements relative to other players, which are very important indicators used to increase players' engagement. Furthermore, the rewards consist only in in-game points, and the main criteria used for ranks candidate challenges involves only the reachability of the game achievements, not the players' overall status, like their leaderboard rank or his level. This is a direct consequence of how the system generates recommendations. They are created and recommended using rules and heuristics rather than patterns in the data, as is typical of RSs.

From the perspective of RSs, it can be observed that the system is a type of Content-Based Filtering RS. This is because it only considers the behavior of individual players and does not take into account any interaction between them. In this RS items are represented by the challenges, while user/item interactions can be represented by the user's performance in that challenge or by the outcome of the challenge. As highlighted by Khoshkangini et al. [2021], also historical data of the users, i.e. their past performance, are an important feature in difficulty calibration of the challenges. Therefore, it comes natural modeling users' profiles as temporally ordered sequences of challenges, with their own features, and build challenges recommendations on these.

To handle the limitations highlighted before and improve the quality of the recommendations, one possible approach might involve either replacing or integrating one or more components within the existing system, such as enhancing the ML module with a more advanced model or incorporating a Generative AI model like ChatGPT for challenge generation. Based on this idea, we have implemented a Large Language Model, a particular type of Deep Learning model, to predict a user's intent based on their behavioral history, in order to explore the potential of this models in the domain of gamification, firstly as a predictor, then as a basis for building recommendation systems for gamified applications - this part will not be deepened but only mentioned in Section 5.3.

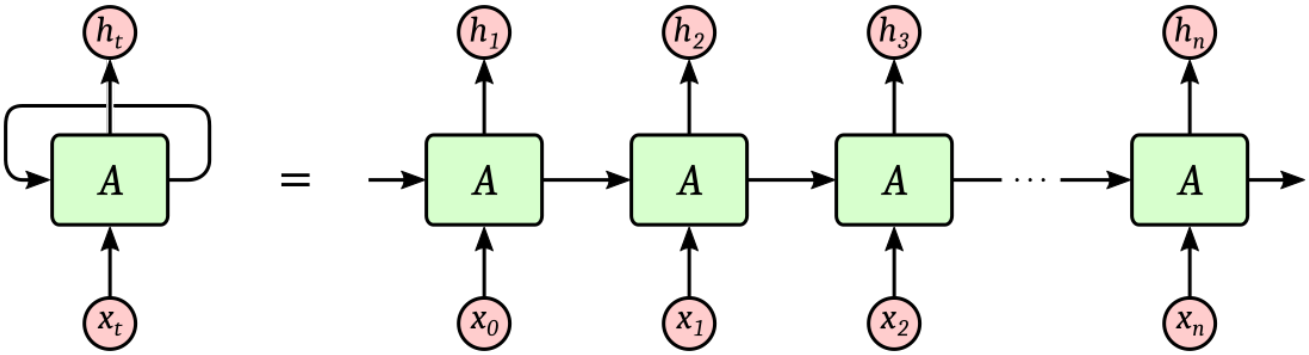

Figure 2.6: Conceptual functioning of an RNN. The output of every unit A depends on the network input and the hidden state of the previous layer. (Image taken from `https://kvitajakub.github.io/2016/04/14/rnn-diagrams/`)

2.3 Sequential recommendations models

As we said in Section 2.2.3, we are required to build recommendations taking into account users' interactions in the form of sequences, so we have to develop a model capable of handling this particular type of input. In these cases we talk about *Sequential recommendations* and the models who produce them are called *Sequential recommender systems*, for which we have a definition took from Li et al. [2020]:

> Sequential recommender systems seek to exploit the order of users' interactions, in order to predict their next action based on the context of what they have done recently.

Traditionally, the most used approach to handle sequences are Markov Chains [Rendle et al., 2010, He and McAuley, 2016], but more recently Recurrent Neural Networks (RNN) [Hidasi et al., 2015], Self Attention and Transformers-based models [Li et al., 2020, Sun et al., 2019] and Graph Neural Network (GNN) [Chang et al., 2021] have proliferated.

Markov Chains are stochastic models that describe sequences of a finite number of possible events, also known as 'states'. The probability of transitioning from one state to another depends on the sequence of states reached over time. If the probability of transitioning depends solely on the current state, it is referred to as a 'first-order Markov chain'. If transitions depend on the current state and a finite number of states before it, it is referred to as a 'higher-order Markov chain'. Markov Chains are a powerful method for analyzing interactions, but they are not sufficient to generate recommendations: Rendle et al. [2010] combined first-order Markov Chains with Matrix Factorization, a classical Recommendation algorithm, while He and McAuley [2016] proposed a model in which similarities between items where calculated and used in combination with high-order Markov Chains.

Recurrent neural networks are a family of artificial neural networks which process sequential data one time step at time. While traditional deep neural networks assume that inputs and outputs are independent of each other, RNNs have connections that form a directed cycle, allowing information to persist, as it can be seen in Figure 2.6. The output at each time step depends on the input at the time step and some information from the previous time step, called *hidden state*. RNNs are a good alternative to Markov Chains, but they are difficult to train and they cannot catch effectively long-term dependencies in sequences. To mitigate this led to the development of more sophisticated architectures like Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), which include specialized

mechanisms to address these issues. An LSTM unit consists of a memory cell and three gates – input gate, forget gate, and output gate. These gates regulate the flow of information into, out of, and within the memory cell, allowing LSTMs to capture and retain long-term dependencies in sequential data. A GRU is another type of recurrent neural network (RNN) architecture, similar to LSTM but with a simpler structure. Instead of three gates, it uses only two gates: the reset gate, which decides how much past information to forget and computes a reset factor based on the previous hidden state and current input, and the update gate, which computes how much of the new information to incorporate in the hidden state. The hidden state itself is also the output of the unit. LSTMs are often better at capturing long-term dependencies, but GRU may be more efficient in situations where less complex models are desired and short-term dependencies are more critical. Hidasi et al. [2015] used a GRU network to build a model for sequential recommendations called GRU-Rec.

Transformers and self-Attention are currently the state of the art in Natural Language Processing (NLP). They were introduced in 2017 by Vaswani et al. as models for text translation. A transformer architecture consists in two main parts, an encoder and a decoder, both composed of multiple layers of multi-head self attention and feedforward neural networks, as visible in Figure 2.7. The encoder processes the input data, while the decoder generates the output sequence. These components can be used together or as standalone models, such as BERT, which is an encoder-only model and GPT, which is instead a decoder-only model.

Similarly to other sequence transduction models, transformers convert the input, text or sequence, in atomic chunks known as *tokens*. Every Transformer has a fixed-size set of recognizable tokens called *vocabulary*. Each token in the vocabulary corresponds to a unique learned embedding vector. Embeddings are the representation of the tokens in an high-dimensional space, where each dimension corresponds to a learned feature or attribute of the language. Unlike RNNs, transformers don't contain any recurrence or convolution to memorize the order of the sequence, so they must inject some information about the relative or absolute position of the tokens in the sequence. To this end, an "absolute positional encodings" are added to the input embeddings before feeding them into the transformer model [Vaswani et al., 2017]. Absolute positional encoding assigns a unique fixed vector of the same dimension of the embeddings to each position in the input sequence. The positional encodings are typically learned during training and can be represented as fixed sinusoidal functions of different frequencies and phases. That is, each dimension of the positional encoding corresponds to a sinusoid. This function was chosen because Vaswani et al. hypothesized it would allow the model to easily learn to attend by relative positions.

Self-attention is a mechanism that allows a transformer model to weight the importance of different tokens in a sequence when processing a specific token of the input. In self attention each token is converted in three vectors through linear transformations of its embeddings: the Key, the Query and the Value. These vectors are packed together as rows of three matrices K, Q and V. Then this formula is applied

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the size of key vectors and softmax is a function that maps every row z of the matrix into a same size row $\sigma(z)$ of values in the range $(0, 1)$ where the sum of these values is 1. Transformers uses multi-head self-attention, which consists in several self-attention layer running in parallel and

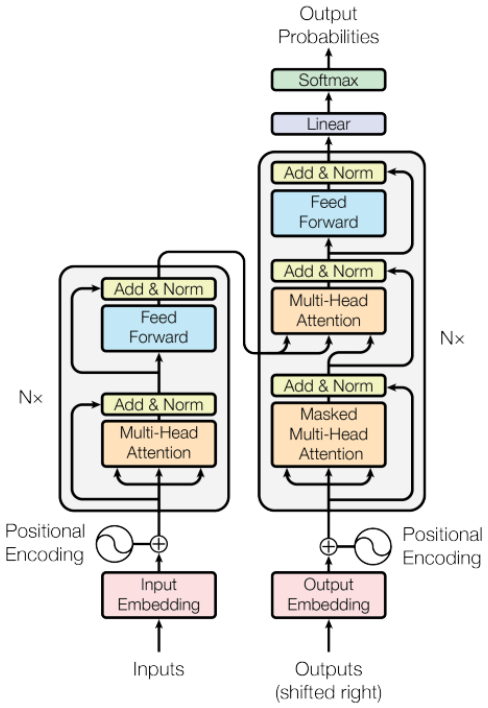

Figure 2.7: The Transformer-model architecture (taken from Vaswani et al. [2017])

concatenated to obtain the output to pass to the Feed-Forward layer. Around each of the sub-layers a normalization layer is employed, in which the output of the sublayer is summed to the input of the sublayer itself.

In addition to the two sub-layers in each encoder layer, in the decoder a third sub-layer, which performs multi-head attention over the output of the encoder stack, is introduced and the self-attention sublayers are modified to prevent positions from attending to subsequent positions. Combining this with offsetting of the output embeddings, it ensures that the prediction for every position depends only on its previous positions. The output of the decoder is passed through a softmax layer, which returns a sequence of probability vectors, in which every value corresponds to a token in the vocabulary. From these probabilities the final sequence can be generated with various algorithms, such as greedy search, which builds the sequence taking always the token with highest associated probability, or bean search, which keeps several hypotheses at each time step and eventually chooses the hypothesis that has the overall highest probability for the entire sequence.

Transformers were born for handling text, but they are rapidly adapted for other forms of inputs like classical sequences. Example of a sequential model based on transformers is Temporal Fusion Transformers [Lim et al., 2021], used in temporal series analysis, while for recommendations BERT4Rec [Sun et al., 2019], SASRec and its variants [Li et al., 2020] was developed.

An alternative approach was proposed by Chang et al. in 2021. Instead of modelize sequences as 1D vectors, it uses directed graphs in which nodes are the items and there is a direct edge from item A to item B whether the user interacted with B immediately after interacting with A. Then these graphs are used as input for a GNN, which predicts the next link in the graph and, consequently, the item to recommend.

2.4 Predict user intent in mobility

Understanding human mobility is a long-standing topic in academic research. More generally, predicting the evolution of human movement allows for a multitude of potential applications, such as predicting the evolution of epidemics [Colizza et al., 2007] or making geo-targeted advertisements Ye et al. [2013]. With the introduction of geolocalization services, such as GPS, the opportunity to study human movement in a qualitatively novel setting is provided [Noulas et al., 2012]. In this context, the sequential model proliferated, thanks to their ability to analyze the historical data about users' movements.

Most existing approaches in mobility prediction focus on next location prediction [Ma and Zhang, 2022]. To predict the next in a chain of points of interest (POI) visited by a user, Gambs et al. [2012] modeled the movements between them as an second-order Markov Chain, but the model can be easily extended to higher orders, and taking the POI with the highest probability with respect the previous two movements of the user. Ye et al. [2013], instead, predicted the next POI using an Hidden Markov Chain (HMM), a variant of classical Markov Chains in which the evolution of observable events depends on internal factors, the *hidden states*, that can't be directly observed. In this work, the authors categorized the POIs in nine categories and they used an HMM to predict the category of the next location, and predict it from this information. In Yan et al. [2021], the authors exploit the behavior patterns of users to extract four types of users and then employ a weighted Markov model to predict the trajectories of the different types of users. In Hsieh et al. [2021], the authors use a time-aware language model based on n-gram models, so-called *T-gram*, to predict human mobility, by exploiting location check-in data. N-gram is a statistical language model that is able to predict the next work in the sequence given the words that precede. Al-Molegi et al. [2018] developed a model called MOP (Move, Attend and Predict), which fuses RNN with Attention to predict locations.

RNN architectures with LSTM units are employed in various mobility prediction contexts. For example, they are used to predict the next location of users in a distributed Multi-Access Edge Computing environment from an automotive perspective [Fattore et al., 2020] and to extract the relations between bike usage and public transport according to real-time and past usage data [Zhang et al., 2018].

Regarding bike sharing, Zhang et al. [2018] employ RNNs with LSTM units to extract relations between bike usage and public transport, and the number of potential bike-sharing users, according to both real-time and past usage data. Similarly, Wang and Kim [2018] employ LSTMs, GRUs and random forests to predict the number of bikes in a certain bike-sharing station.

Regarding next trip prediction, by taking advantage of transit smart card records Zhao et al. [2018] develop a methodology to predict whether users will travel on a particular day and what attributes the trip will have. The authors employ the Bayesian n-gram model to retrieve a probability distribution of the next trip.

2.5 Large Language Models for Recommendations Systems

In recent years even more sophisticated approaches to recommendations in various fields are needed. In this context Large Language Models (LLMs) have emerged as powerful tools for building RSs.

A Large Language Model is a Machine Learning model, based on the *transformer* architecture, designed to understand and generate human-like language. The term "large" implies that the model

has a huge number of trainable parameters in the order of magnitude of a million - the largest models can reach tens of billions -, and it was pre-trained on massive datasets containing a wide range of text from various online sources.

LLMs are known for their versatility. They can effectively handle a wide array of Natural Language Processing (NLP) tasks like text completion, summarization, translation, question answering, and even creative writing. Recently they were used also for tasks different from NLP, like common reasoning [Brown et al., 2020].

One way to extend LLMs capabilities without starting from scratch is *fine-tuning*. Fine-tuning, in the context of LLMs, is a process that comes after the initial pre-training phase. It involves further training the model on a smaller, task-specific dataset. During fine-tuning, the model adjusts its parameters based on the new task-specific dataset. The goal is to refine the model's understanding and performance for the particular application.

LLMs, using natural language as input, provides a simple and flexible way to represent user and item features, contextual information and interactions. In addition, LLMs have shown good ability in dealing with some typical problems of recommendation systems, such as cold start [Hou et al., 2023], explainability, and sparsity [Gao et al., 2023].

In Gao et al. [2023], Zhang et al. [2023b], and Yue et al. [2023], LLMs are used to generate recommendations about items (such as movies) with explanations and reasons for the suggestions given, by converting user information and interactions into prompts. In Gao et al. [2023], ChatGPT is used directly, while in the other two the chosen models (T5 and LLaMA) were fine-tuned before use.

A completely different approach consists of using LLMs as feature generators and using the obtained data with classic RecSys models [Di Palma, 2023]. For example, Acharya et al. [2023] used Alpaca in combination with BERT. In this setup, Alpaca is used to generate textual descriptions of items. These are then given as input to BERT, which computes numerical embeddings that a GRU4Rec model uses to generate the recommendations.

Another promising approach is proposed by Wang et al. [2023]. They used LLMs to construct personalized reasoning graphs that represent the user as a whole (by relying on static features, interactions, etc.) in order to feed them to a Graph Neural Network (GNN) and get the next item.

In general, there are various methods and approaches for recommendation systems based on LLMs and their training methods, including fine-tuning, prompt distillation [Li et al., 2023], and adapters. Fan et al. [2023] conducted a systematic literature review on these types of RSs.

II

Experiments and Results

3

Methodology

This chapter focuses on the methodologies and tools adopted in our experiments. Initially, Section 3.1 will clarify the problem statement and the origins of the data. Subsequently, Section 3.2 will outline the comparative baselines. The construction and preprocessing of the dataset will be presented in Section 3.3, followed by an exploratory analysis of the same. The text prompting technique for the model will be explained in Section 3.4. Concluding the chapter, Sections 3.5 and 3.6 will respectively detail the procedures for training and inference, and the metrics used for evaluation.

3.1 Problem Formulation

Our study focuses on analyzing data from mobility challenges assigned during P&G campaigns promoted during the 2023 edition of the Air Break project (see Section 2.2). In more detail, we want to be able to predict which is the next challenge for each user and recommend it. We adopted the workflow in Figure 3.1. We start from raw data of Play&Go Gamification Engine and from them we builded a sequential data set using a windowing schema. Then, the sequential data is converted into text prompts in order to fine-tune a pre-trained LLM Text-to-Text Transfer Transformer model (T5) and some baseline models. We will describe the model in Section 3.1.1. To ensure our results are conclusive and free from random variability or bias, we adopt two different approaches to the cross validation method in our experimental setup.

The first one is the **k-fold cross validation**, where the prompts are partitioned in k subsets and, at fold i, the i-th subset is used as test set, and the remaining data is used as train set. In our case we use a 5-fold cross validation. The other method is the **time series split cross validation** process, which is specifically designed for evaluating the performance of models on time-dependent data. The time series split involves splitting the dataset into multiple folds, where each fold consists of a temporal contiguous block of data points.

For each mobility challenge, we collect 4 data samples from each user's history to create sequences of length 5. Our goal is to predict the outcome of the last challenge in each sequence. The user's recorded data reflects the interaction between their historical challenges and the newly introduced challenge. Following our time series split validation method, we labeled each challenge in sequences from 0 to 4. At fold i, we used challenges 0 to $i-1$ of each sequence for training and the sequence up to i for testing.

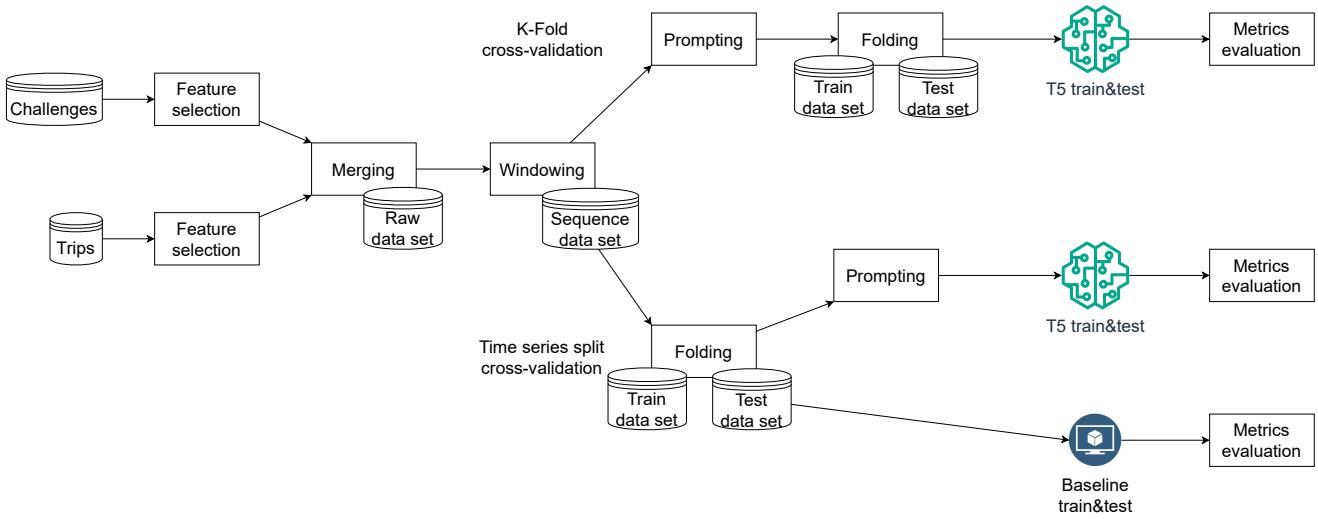

Figure 3.1: Our experimental workflow.

In Figure 3.2 there is an example of how the train and test instances are generated. The effectiveness of our models is then assessed based on the average results of these test sets.

3.1.1 Text-to-Text Transfer Transformer model (T5)

T5 is a pre-trained LLM, i.e. an encoder-decoder transformer, introduced in "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" and developed by Raffel et al. in 2020 at Google. As the name and its authors suggest, it was developed as a unified framework that converts every language problem into a text-to-text format. This makes it a very flexible model and suitable for fine-tuning on various tasks, not only NLP ones [Roitero et al., 2023], so the model can be fine-tuned on multiple task simultaneously or in different moments, such as translation and summarization, simply adding a prefix on inputs that identifies the task to resolve, as can be seen in Figure 3.3.

T5 in its base implementation closely follows the original proposed form [Vaswani et al., 2017] but with some difference. T5 uses a simplified version of the layer normalization where only the sublayer output is rescaled, no additive bias is applied and only after normalization the output is summed with the input. While the original Transformer used absolute position embeddings, T5 uses a simplified form of where each position embedding, instead of being a vector, is simply a scalar value that is added during the computation of the attention weights Raffel et al. [2020]. We used the T5 base implementation made by HuggingFace[1] which has 6 encoding layers and 6 decoding layers, the encoder layer size, i.e. the maximum number of input tokens, is 512, the feedforwards sublayers have size 2048, the vocabulary is large 32128, the key and value vectors have size 64.

The model was pre-trained on a multi-task mixture of unsupervised and supervised tasks. The unsupervised training consists essentially in masked language modeling, which means that the model is fed with texts where some part of it are elided and it learns to fill the elided parts, as can be seen in figure 3.4. The dataset used in this phase is the Colossal Clean Crawled Corpus (C4) [Raffel et al., 2020], a cleaned version of Common Crawl created by the same authors of T5 to pre-train this type of model. Common Crawl[2] is a publicly-available web archive that provides "web extracted text" by

[1] `https://huggingface.co/t5-base`, `https://huggingface.co/docs/transformers/model_doc/t5`
[2] `https://commoncrawl.org/`

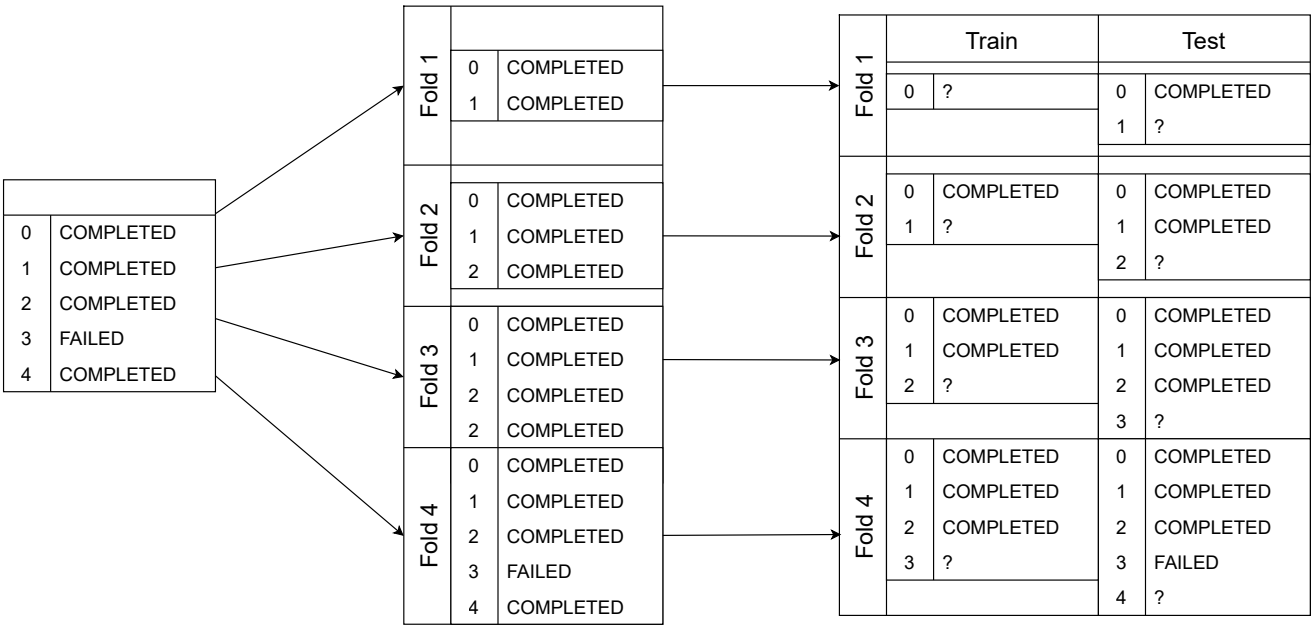

Figure 3.2: Generation of train and test instances from a generic sequence using time-split. For each instance the query is made masking the outcome of the last challenge with '?'.

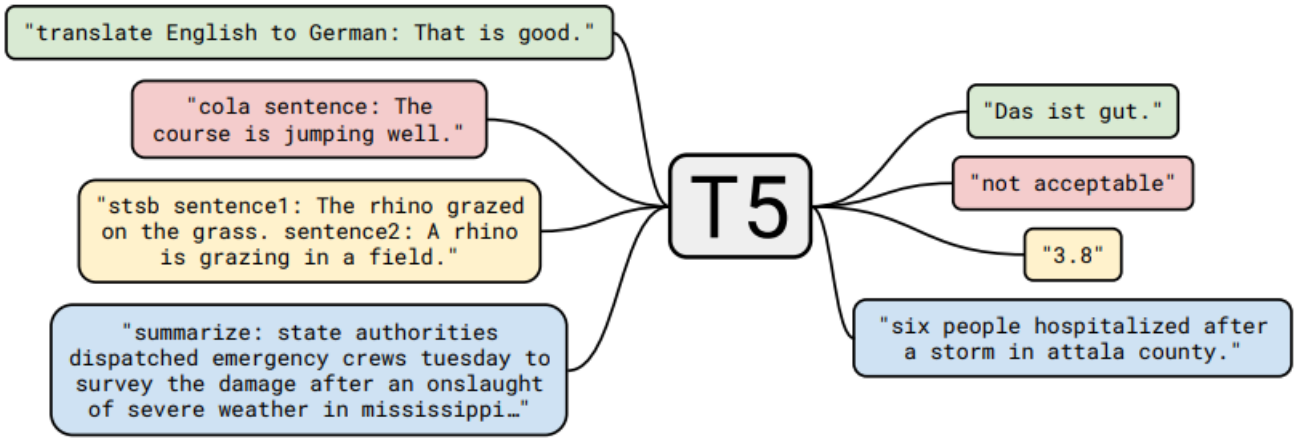

Figure 3.3: A diagram of T5 framework (taken from Raffel et al. [2020]). In this image T5 is used for translation, linguistic acceptability evaluation, sentiment analysis and summarization. Every considered task is cast as feeding the model text as input and training it to generate some target text. This allows the use of the same model, loss function, hyperparameters, etc. across a diverse set of tasks.

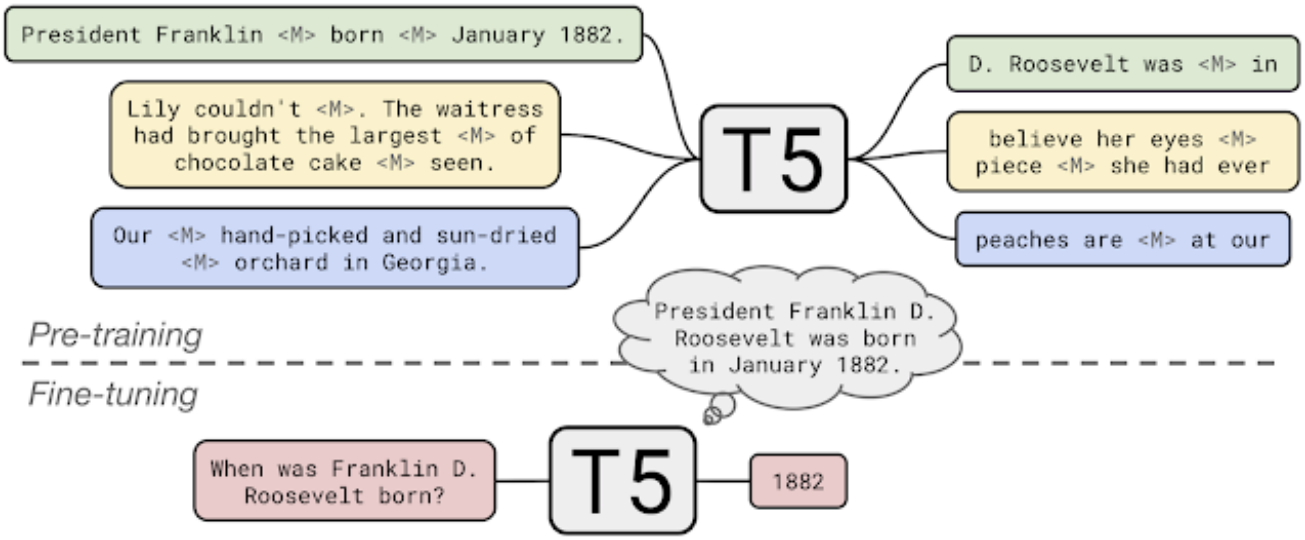

Figure 3.4: Schema of T5 pre-training and fine-tuning, taken from `https://blog.research.google/2020/02/exploring-transfer-learning-with-t5.html`. During pre-training, T5 learns to fill in dropped-out spans of text (denoted by $< M >$) from documents. During fine-tuning, the model uses the learned "knowledge" to resolve the required task.

removing markup and other non-text content from the scraped HTML files. The cleaning process involved deduplication, discarding incomplete sentences, and removing offensive or noisy content. Fine-tuning is, instead, a classical supervised learning process, where an input text is mapped to an output text and the objective is to learn to reproduce the ground truth. The base implementation was fine-tuned and validated on various downstream NLP tasks, including sentence acceptability judgment, sentiment analysis, natural language inference, sentence completion, word sense disambiguation and question answering.

3.2 Baseline

In our research, we adapt and evaluate the following deep learning (DL) architectures, each being notably efficient for time series analysis in diverse scenarios.

Firstly, as a baseline we integrate a simple Multi Layer Perceptron (MLP) with Rectified Linear Unit (ReLU) as activation function.

Another framework in our set of baselines is the Neural Hierarchical Interpolation for Time Series Forecasting (N-HiTS) [Challu et al., 2022]. It is an extension of NBeats model [Oreshkin et al., 2019] that employs a combination of backward and forward residual links along with a deep stack of fully connected layers, operating on input - look-back period - and output - forecast period - segments for time series predictions tasks.

Subsequently, we integrate the Long Short-Term Memory (LSTM) network [Yu et al., 2019], known for its sequence-to-sequence predictive capacity. The LSTM's design allows it to retain information over varying periods, making it particularly adept for analyzing time series data in mobility contexts, where intervals between significant events can vary on per-instance bases.

We also employ the Gated Recurrent Unit (GRU) network [Dey and Salem, 2017], an evolution of the LSTM, engineered to address the vanishing gradient issue found in the former models. The GRU

utilizes two gates - the update and the reset gate - to modulate the flow of information deciding which information to use and which to discard for the generation of the output. In more detail, the update gate is crucial in deciding how much past data is needed for future predictions, whereas the reset gate determines what proportion of past data should be ignored.

Further, we incorporate the Temporal Fusion Transformer (TFT), a model developed by Google [Lim et al., 2021], which uses an attention mechanism. This architecture is composed of various elements such as a temporal multi-head attention block for detecting long-term trends, LSTM sequence-to-sequence encoders/decoders for short-term patterns, and Gated Network blocks to filter out the less relevant inputs.

Together, these models form an extensive suite of advanced deep learning methods for time series analysis. We pair these with our newly developed Transformer-based approach, tailored for predicting outcomes in mobility challenge scenarios.

3.3 Dataset Construction and Analysis

The dataset used in our experiments is constructed from Play&Go log data collected during the 2023 edition of the Air Break project. In particular we use data about users' movements and data about assigned challenges during the campaigns. We aim to cross-reference this data in order to build a coherent dataset about users behaviors during Play&Go campaigns.

The building process was composed by four main phases, also visible in Figure 3.1:

1. a *preprocessing* phase, where we take the raw data in order to extract and format the needed information,

2. a *merging* phase, where we merge the data to obtain the final dataset,

3. a *windowing* phase, where we take the obtained dataset to build the sequences used as inputs for the models,

4. a *prompting*, where we take the sequences and we convert them into textual prompts.

The last two phases will be described in-depth in Section 3.4.

3.3.1 Preprocessing and feature selection

In this phase we aim to take the raw data from Play&Go logs, filter and format them in a table-like structure.

The data is extracted from the Play&Go database as text files. Each row represents an in-game event described by a JSON string. Although the files themselves are not in JSON format, we can convert each JSON string into a Python dictionary by reading the files line by line. The resulting lists can then be converted into Pandas data frames for further manipulation.

In the next two paragraphs we will briefly describe the structure of the logs file, the selected features and any transformation applied on them.

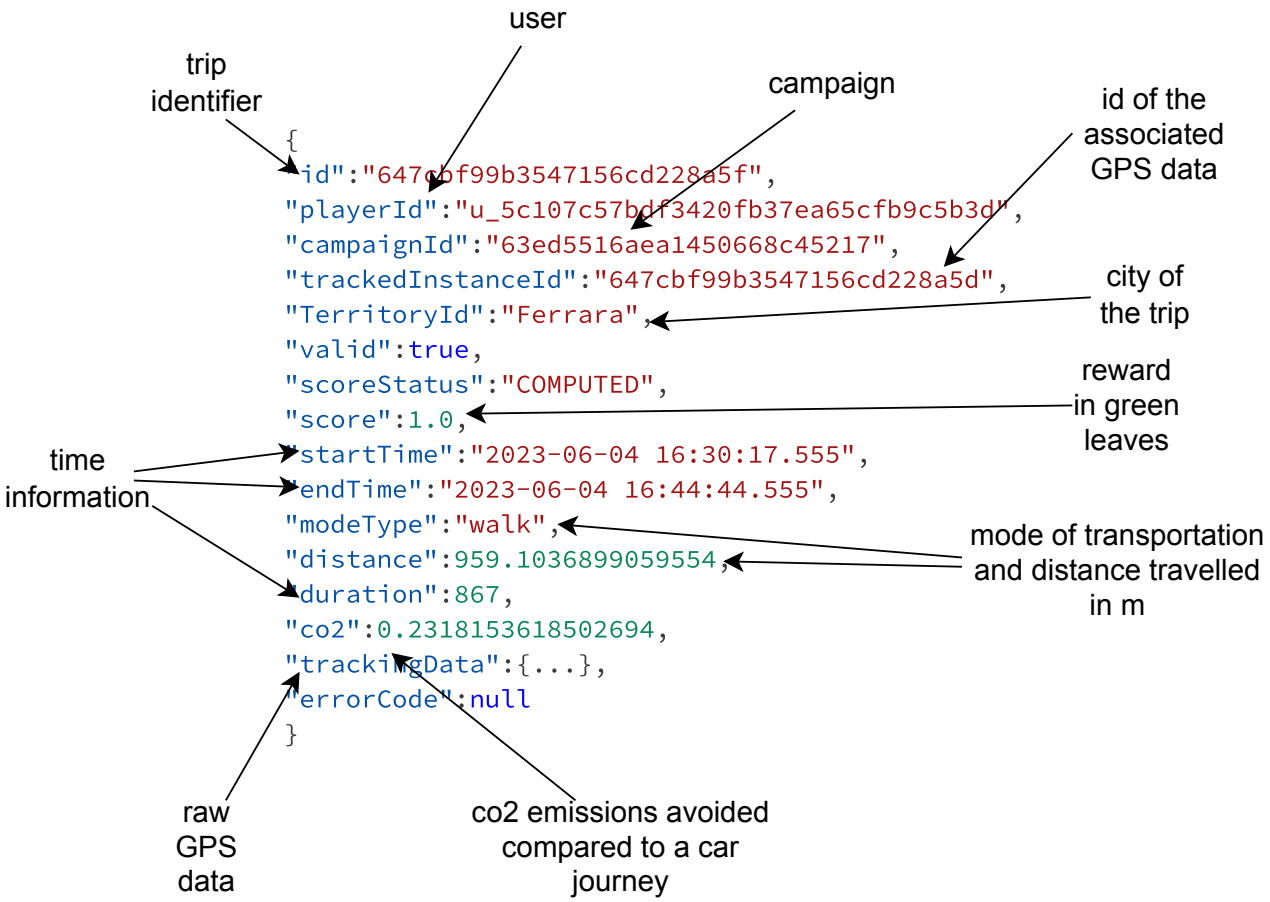

Figure 3.5: Example of a trip log.

Trips

As trip we mean a single movement of the user between two points for a certain distance, greater than a minimum one, on foot or by some allowed means (bicycle, bus, train or with carsharing). The trips are collected and validated using GPS data.

In the Figure 3.5 there is an example of a trip log with a brief explanation of the main fields. From trips log we select this fields: `playerId`, `TerritoryId`, `startTime`, from which we extract the `year` and the `week` number, `modeType` and `distance`. Other fields are either meaningless, useless, or contain redundant information.

Therefore, obtained trips data, we aim to create a summary where for each player and week we have the total distance traveled for each mode of transportation, e.g. the player A during the week 16 of has traveled 1 Km by walking, 2 Km by bike, 0 Km by train, *etc.*.

For doing this we follow these steps.

1. The trips data frame is grouped by `playerId`, `week`, `year`, `TerritoryId` and `modeType`. The distance columns are aggregated by summing the values and converting to kilometers.

2. One-hot encoding is performed on `modeType`.

3. One-hot encoded columns are multiplied by `distance`. In this way the distance value is transferred from the original column to the column representing the correspondent mode of transportation.

4. The data frame is grouped by `playerId`, `week`, `year` and `TerritoryId` and the remaining columns aggregating adding them.

Lastly, 1 week is subtracted from time information. In this way we will treat the trips from the previous week as context for the current week during the merging phase (see Section Merging).

Challenges

For challenge assignments the log has the structure shown in Figure 3.6. In this case the process to get the required features follows these steps.

1. As the logs also contain data on surveys sent to users during campaigns, we remove the rows with *modelName* field in `concept` equal to `"survey"`.

2. To make the dictionaries compatible with Pandas, we have to flat them. Flat means to convert a nested dictionary into a flat dictionary, where all key-value pairs are at the top level.

3. We merge flattened dictionary in a unique Pandas data frame and we remove any row with all columns equal to `NaN`.

4. We select these columns from the data frame: `start`, `target`, `periodTarget`, `counterName`, `state`. Other columns are meaningless or contain redundant information (e.g. the `periodName` field should contain measurement unit for `periodTarget`, but it has always the same value for all rows).

5. From `start` we extract `year` and `week` number, in the same way as seen for trips.

6. We apply some transformation on selected data: `target` and `periodTarget` are converted into integer values and we substituted any whitespace in `counterName` with an underscore.

7. As the obtained data frame also contains ongoing challenges we removed rows with `state` not equal to ‘‘`COMPLETED"` or ‘‘`FAILED"`.

8. Lastly, the data frame is ordered by `playerId` and `start`. In this way we have all challenge assignments chronologically ordered for each user.

3.3.2 Merging

In this phase we take the trip and challenges tables created during the previous phase to merge them and create the dataset for the prompting phase (see Section 3.4) and it is created following these steps.

1. From trips data frame we create a new data frame with unique `playerId` and `TerritoryId` pairs, then `TerritoryId` column is dropped.

2. A **right join** is executed between the trips and challenges data frames using the `playerId`, `week`, and `year` as the joining keys. Since we subtracted one week, the trips summary from the previous week is assigned as context for each challenge assignment.

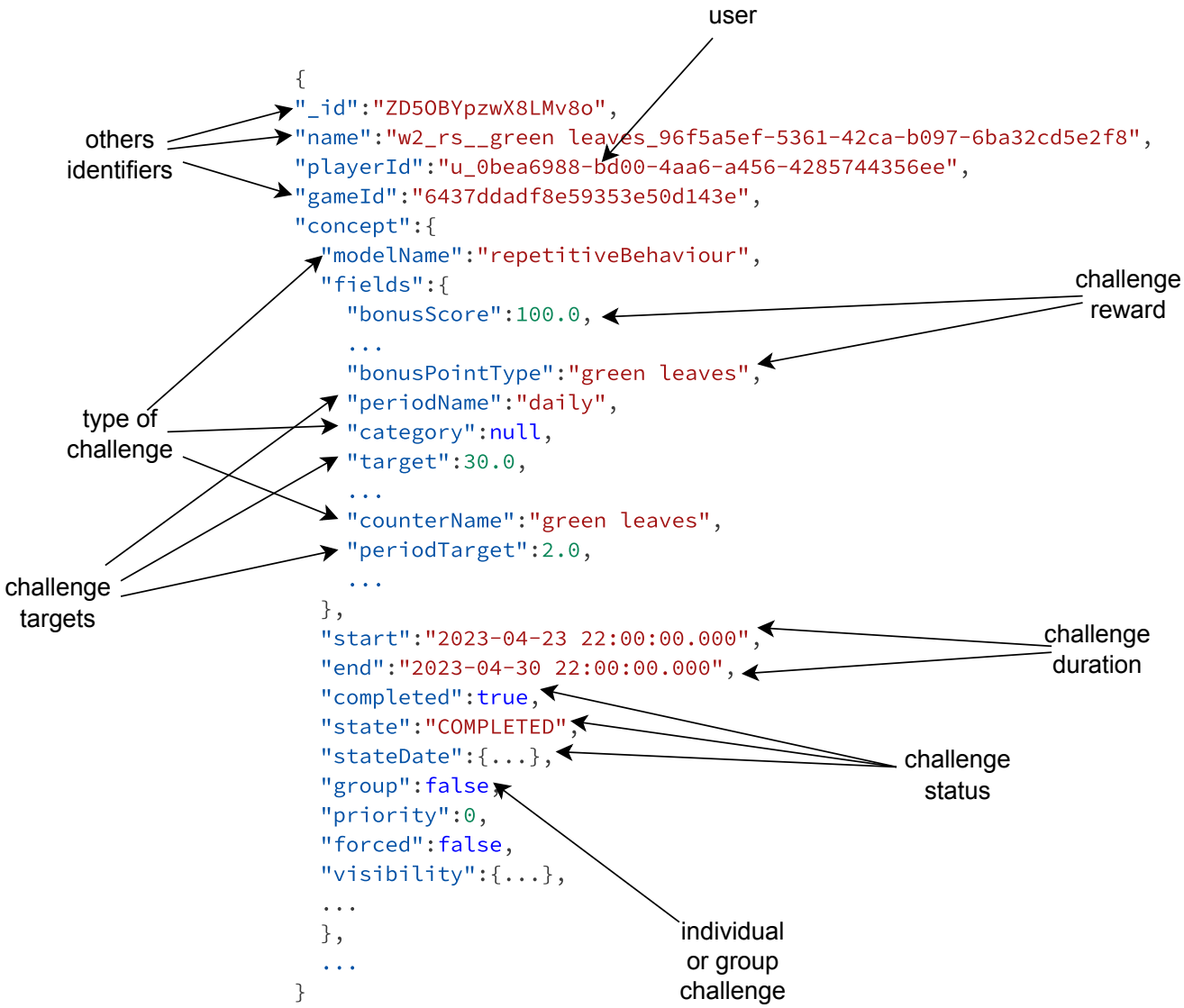

Figure 3.6: Example of a challenge log.

3. Therefore, an **inner join** is performed between the data frame obtained and the one created at point 1.

The final dataset comprises 6016 sustainable mobility challenges collected between April and September 2023 in Ferrara, Italy. Each row of the dataset contains 13 features, 7 numerical (bike, bus, car, train, walk, target, periodTarget), 2 temporal (week and year), 4 categorical (playerId, TerritoryId, counterName and state) as described in Table 3.1.

3.3.3 Dataset Exploration

In this Section we will make an exploratory data analysis of the dataset obtained following the procedures described in section 3.3. Another goal of this chapter is to extract and analyze information about the users' behavior during the game.

All plots described in this analysis are identified by the capital letter alongside the title.

The dataset seems to be slightly unbalanced in relation to the response variable, i.e. `state`, as shown in the barplot (A). Therefore, our model must be able to handle this imbalance.

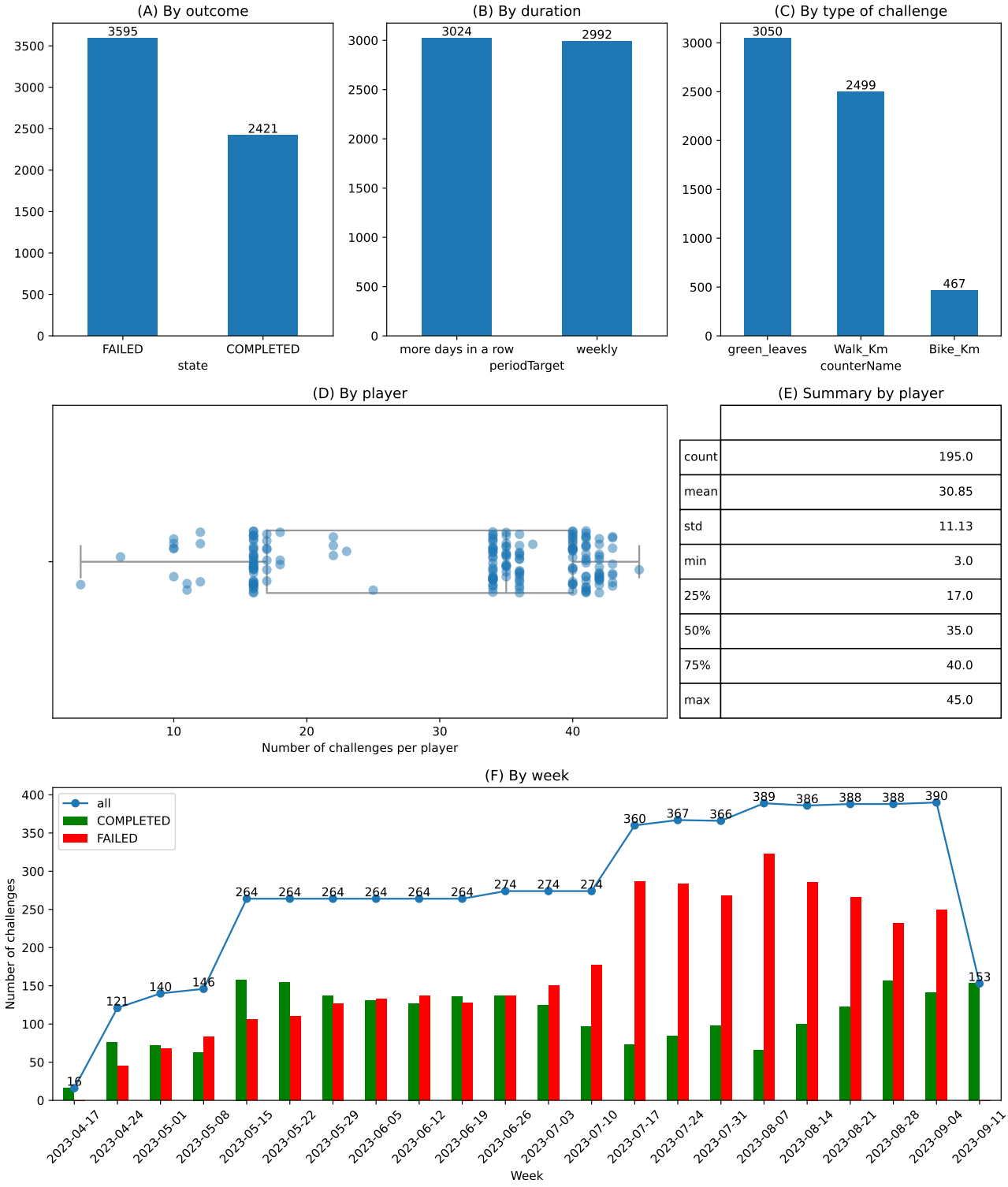

Figure 3.7: Explorative analysis of the dataset. The plots (A), (B) and (C) show the distribution of challenges grouped by three categorical features: `state`, `periodTarget` and `counterName`. The boxplot (D) and the summary table E represents the distribution of assigned challenges for each player. The plot (F) shows the number of challenges for every considered week, the bars under the line represents the in-week distribution of completed and failed challenges.

Feature	Description
`playerId`	the identifier of the user in the system
`TerritoryId`	the city where the user is located when the challenge was assigned. Even if this feature is always equal to 'Ferrara' in the dataset, it was retained to be used in future works
`week`	the week number (between 1 and 53)
`year`	the year of the challenge assignment to the user
`bike`	the total distance (in Km) traveled by the user using the vehicle indicated by the column during the *previous* week
`bus`	
`car`	
`train`	
`walk`	
`counterName`	the type of challenge (walk, bike or green leaves earning)
`target`	the target value to reach of the challenge (in Km or green leaves)
`periodTarget`	the duration of the challenge. If 0 means that the target value must be reached during the week, else if greater than 0 means that the target value must be reached for multiple days in a row during the week (e.g. $[green\_leaves, 30, 2]$ means that the user have to earns 30 green leaves every day for 2 consecutive days)
`state`	the outcome of the challenge (COMPLETED or FAILED). This is the value to be predicted by the model.

Table 3.1: Features of the dataset.

Regarding the barplot (B) and (C), we observe good balancing. If we sum the frequencies of *'Walk_Km'* with *'Bike_Km'*, we obtain almost the same number of *'green_leaves'* challenges. This is a consequence of how challenges are assigned to users. In almost all weeks considered, the existing system assigned two challenges to each user. The first challenge was a simple weekly task, such as walking or biking. The second challenge involved collecting green leaves for multiple days. The barplot (B) is obtained partitioning the dataset in samples where `periodTarget > 0` and samples where `periodTarget = 0`.

The boxplot (D) and table (E) display the number of challenges assigned to each player. The blue points in the boxplot represent individual measures. Four clusters of points are clearly observable: the first around 10, the second around the first quartile, the third around the mean, and the fourth between the third quartile and the maximum. Assuming that the oldest users are those with the most challenges, we can conclude with a good approximation that each cluster corresponds to a group of users that subscribed to the system during the same period.

The plot (F) represents the distribution of challenges among weeks. On the x-axis there are the single weeks, represented by the Monday of the week in question. The marked blue line is the trend of challenges. The barplots under it represent the distribution of completed - the green bars - and failed - the red bars - challenges for each week.

This plot indirectly confirms the hypothesis regarding clusters of players. The trend line shows 2-3 periods where the number of challenges assigned remains constant, interrupted by four weeks where

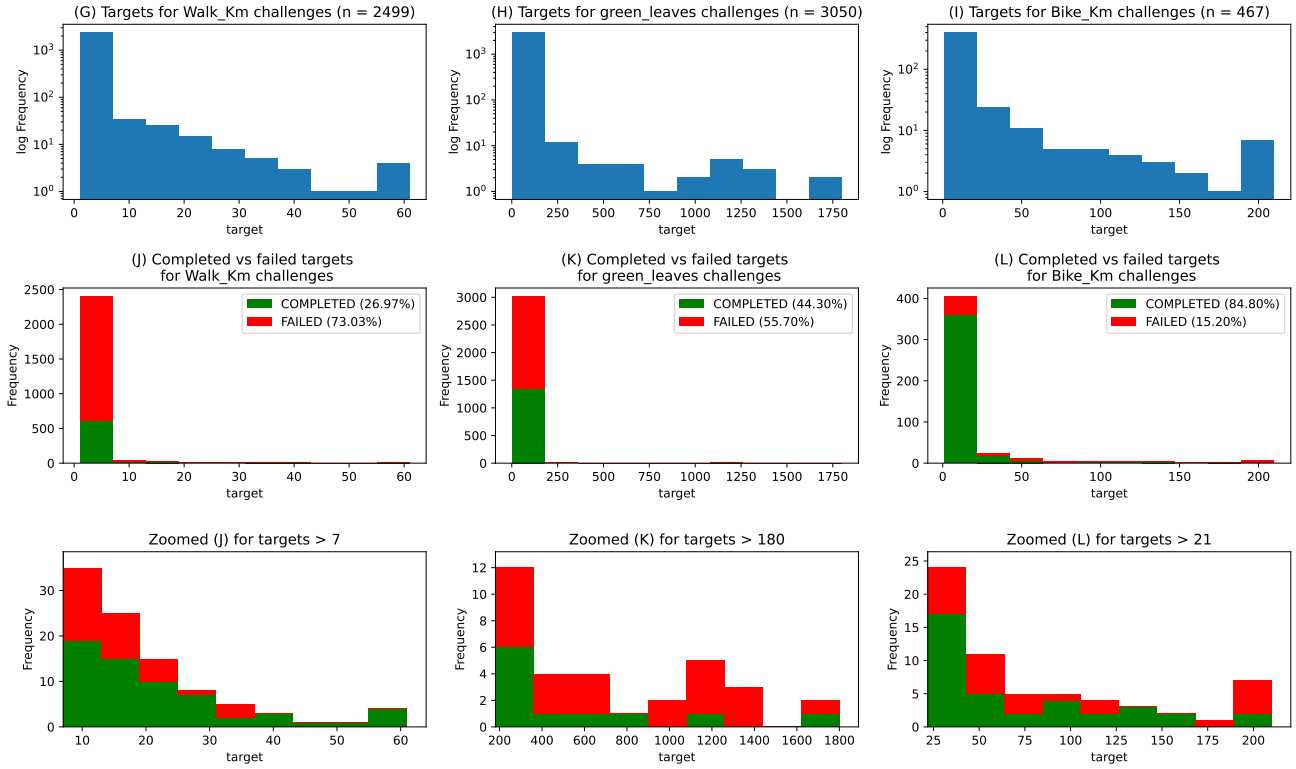

Figure 3.8: Distribution of challenges' targets for each counterName.

the number increases abruptly. These weeks are: the week between April 24th and May 1st, the week between May 8th and 15th, the week between July 10th and 17th, and the week between August 7th and 14th.

The decrease in the number of challenges last week is attributed to the fact that at the time of data extraction, not all challenges had been completed and were therefore excluded from the dataset.

When observing the distribution of challenges throughout the week, the ratio of failed to completed challenges remains consistent during the first period of time. However, there is a strong bias towards failures from the start of July to the end of August. This can be easily explained by the fact that these are the two months in which more people go on holiday and may underperform or abandon the game momentarily.

Histograms (G), (H) and (I) in figure 3.8 representing the distribution of challenge targets over the dataset, grouped by their `counterName`. The x-axis displays the target values, while the y-axis displays the frequency of each value in logarithmic scale to better highlight the outliers. In all cases, a highly right-skewed distribution is observed, where the lowest values cover almost all frequencies, but with higher values retaining a not negligible fraction of the data set. This fragment describes an important indicator of user behavior during the game. It suggests that the majority of users maintain a consistent level of performance, while a small group of highly active users show a high level of improvement over time, so the system tends to assign to them more difficult challenges.

Histograms (J), (K), and (L) represent the data using a linear scale. The bars in the histograms are divided into completed (green) and failed (red) challenges. The legends display the percentages of completed and failed challenges for the counterName in question, rather than the total number of challenges. The histograms in the third row present the same data as the histogram above them, but

only for targets that exceed the values represented by the first bars. The plots indicate that users have completed a significant portion of the assigned bike challenges. However, these challenges are less frequently assigned and have a heavier tail distribution, suggesting that bicycle is a popular mode of transportation for this sample of users. Conversely, the system assigns many walk challenges, but users fail most of them. In both cases, these considerations could also be valid for higher targets. However, due to the small sample size, it is not possible to draw satisfactory conclusions. Concerning the challenges where users have to collect green leaves, the ratio of completed to failed attempts is slightly biased towards failures. Unfortunately, we do not have enough information to make any hypotheses about this observed pattern.

To improve the analysis of users' behaviors, we examine the dataset described in Section Trips, which includes the actual performance of users as shown in Figure 3.9. The bar chart (M) displays the total number of trips completed by users, grouped by their mode of transportation. Note that the original dataset also included trips referring to cities other than Ferrara, so they were removed. It is clear that walking and cycling trips make up the majority of the dataset, with a large prevalence of cycling ones. Therefore, we plotted histograms (N) and (Q) to show the distances covered by each trip in the two categories, as well as histograms (O) and (R) to show the total distances covered by users each week, and histograms (P) and (S) to show the number of trips made each week with the considered modality. For each histogram, we highlighted the average values. Normalizing the distances, for instance, through Min-Max Normalization, reveals that although users cover a greater distance on average by walking than by cycling for single trips, weekly users make more trips by bike than by walking (7.02 trips per week compared to 5.58). Additionally, the covered distance is greater, thus confirming our initial hypothesis.

3.4 Windowing and Prompting

For converting dataset rows into sequences and sequences into text prompts for T5 we follow these steps.

1. Each row is converted in a Python dictionary following this schema:

   - City: the `TerritoryId` values
   - Challenge: information about the challenge using a string template (walk for X km/bike for X km/collect X green leaves). If the `periodTarget` is greater than 0 the suffix "for Y days in a row" is added;
   - Trips: a dictionary containing `bike`, `bus`, `car`, `train` and `walk` values normalized using min-max normalization (for improve stability),
   - Outcome: the state value.

   It should be noted that no user or time-related information is included in order to prevent the model from "learning from the future" and ensure that each sequence is treated independently.

2. The rows are grouped by playerId and aggregates to create the full history of the user.

3. From previous histories, we generate sequences using a sliding window schema. The schema progresses as follows: [0], [0,1], [0,1,2], [0,1,2,3], [0,1,2,3,4], [1,2,3,4,5], and so on until all the user's

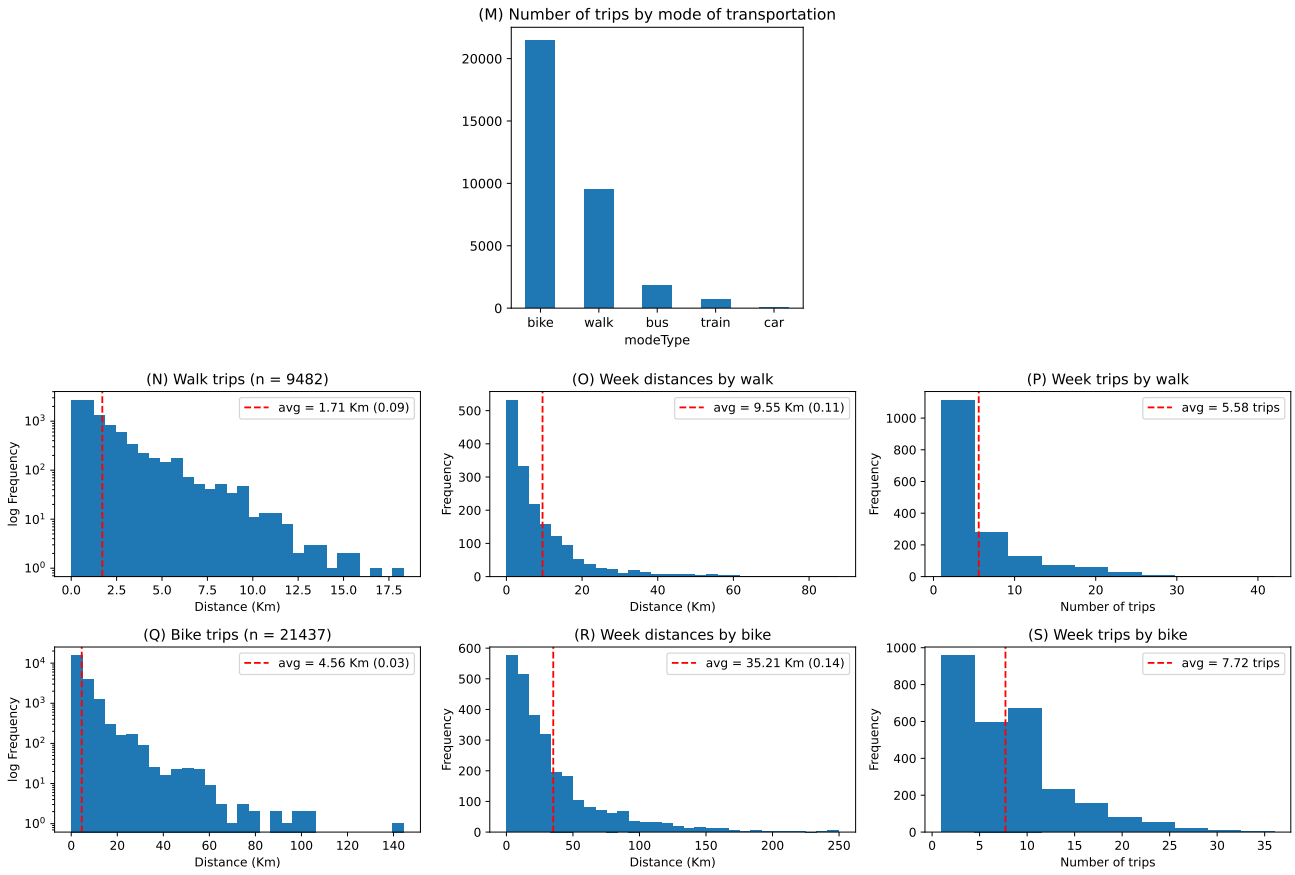

Figure 3.9: Analisys of trips dataset. The barplot (M) represents how many trips users made for each mode of transportation. Histograms (N) and (Q) show the distribution of covered distances of each trip by bike and by walking in logarithmic scale. Histograms (O), (R), (P) and (S) aggregate data about trips for each week and for each user by summing the distances and counting the number of trips completed. Red lines represent the average value. In legends the value of the average value in Km and normalized in scale $[0, 1]$.

history is covered. Keeping sequences limited to a maximum of five elements is necessary due to the 512 token limit for T5 inputs. Each element within the sequences is assigned a sequence identifier (a unique cumulative number for each sequence) and an in-sequence identifier (a cumulative number ranging from 0 to 4).

4. We generate the folds for time splitting validation in the following way: for an integer x ranging from 1 to 4, we select the rows with in-sequence identification that is less than or equal to x. Afterward, we create a training set, which excludes the last row of each sequence, while the original sequences constitute the test set.

5. To create the folds for cross-validation, we divide the sequence identifiers into 5 subsets. At fold i, we choose all the sequences with an identifier within the i-th subset to generate the test set. The remaining sequences are then used as the training set.

6. To generate the final prompts for both the training and testing stages, we aggregate the sequences into Python lists using their sequence identifier. We then mask the outcome of the last challenge and extract it as ground truth. Finally, we convert the list into a string using the Python `str` function.

Here there is an example of a (formatted) prompt:

```
"[
    {
        'City': 'Ferrara',
        'Challenge': 'walk for 1 km',
        'Trips': {'bike': 0.0, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.0},
        'Outcome': 'FAILED'
    },
    {
        'City': 'Ferrara',
        'Challenge': 'collect 30 leaves for 2 days in a row',
        'Trips': {'bike': 0.0, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.0},
        'Outcome': '?'
    }
]"
```

3.5 Training and inference

We constructed our model via Pytorch and the `T5-base` HuggingFace implementation. The weights of the model are initialized with the pre-trained ones of the original T5 model. The model is trained for 3 epochs and the textual input is directly feeded into it (no prefixes are added). As objective we use multi-class cross-entropy loss function, defined as

$$\mathcal{L} = -\frac{1}{B}\sum_{b=1}^{B}\sum_{k=1}^{|V|} y_k^b \log(\hat{y}_k^b)$$

where B is the batch size, b the current batch, $|V|$ the model vocabulary size, y the true token to be predicted, $\hat{y}$ the output token probability distribution over the vocabulary at each time step.

For inference, the output text is generated using greedy decoding, with a limit on the number of tokens to 32. It has been experimentally observed that our fine-tuned model consistently produces the correct string of either 'COMPLETED' or 'FAILED' without the need for any specific constraints or output preprocessing.[3].

To train the deep learning baselines (see Section 3.2) we use NeuralForecast, an open source framework developed by Nixtla, which includes an out-of-the-box ecosystem for train, evaluate and deploy a lot of pre-implemented time-series models like LSTM, DeepAR, NBeats, *etc.*

All experiments are executed in a Linux server running Ubuntu 18.04.3 LTS, inside a Conda environment with Python 3.10.13, PyTorch 2.0.1 (compiled for Cuda 11.7), HuggingFace transformers 4.35, NeuralForecast 1.6.4 and Pandas 2.0.3.

[3]For training and inference we use the default implementation of the HuggingFace library, which include the loss function shown in this section

3.6 Metrics

For assessing the effectiveness of our model in the context of mobility challenges, we employ a range of metrics designed to measure various aspects of model performance. We adapt the standard terminology for classification to our scenario as follows:

- TP (True Positives): instances correctly identified as 'COMPLETED' to a specific mobility challenge.
- TN (True Negatives): instances correctly identified as 'FAILED' to any mobility challenge.
- FP (False Positives): instances mis-classified as 'COMPLETED' to one of the mobility challenges when they were not.
- FN (False Negatives): instances where a completed mobility challenge is present but it was presented as failed.

The following specific metrics are computed:

- Accuracy, defined as

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}.$$

- Precision, defined as

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

- Recall, defined as

$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

- F1-Score, defined as

$$\text{F1} = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}.$$

Considering that we only have two possible classes to predict, we will only use the binary version of these four metrics, where 'COMPLETED' is considered the True class and 'FAILED' is considered the False class.

4

Experimental results

In this chapter, we show the results of our experiments. They were conducted using the research questions presented in Section 4.1. The results are presented in Section 4.2, Section 4.3, and Section 4.4. Finally, Section 4.5 presents an analysis of the predictions obtained by our model during the last fold of the time-splitting cross-validation.

4.1 Research questions

In our experiments we executed the two methods of cross-validation calculating and collecting the metrics for each fold. These metrics are then aggregated and analyzed, focusing on the following questions:

RQ1 Which cross-validation approach to fine-tuning our model gives us the better results?

RQ2 Does our model outperform the baseline?

RQ3 Does sequence length matter in predictions?

4.2 RQ1: Comparison between k-fold and time series split cross validation

Table 4.1 illustrates the computed effective metrics, derived by calculating the mean and maximum scores across each fold in the two proposed cross-validation methods: time series split and k-fold, which corresponds to two different approaches to train and make inferences with the model. The first approach, represented by the k-fold method, is more similar to classical use of ML models and consists in training the model with some prompts and making inference on completely different prompts. The second approach, represented by time series split, is more similar to the use of RNN models and consists in training the model with some prompts and making inference on prompts which are the continuation of the prompts used for training.

Figure 4.1 shows the same metrics through box plots. The x-axes of the box plots represent the two methods, while the y-axes display the corresponding scores. The colored points within the box plots denote individual measurements. It is evident from the table and the plot that the two methods have similar performance, with the only difference being that time-splitting returns the best maximal

Method	Statistic	Accuracy	Precision	Recall	F1
k-fold	max	0.929	0.898	**0.946**	0.914
time-split	max	**0.932**	**0.899**	0.938	**0.917**
k-fold	mean	**0.923**	**0.880**	**0.937**	**0.908**
time-split	mean	0.890	0.856	0.875	0.865

Table 4.1: Mean and maximum of effectiveness metrics computed over folds of the two cross-validation methods. Highest values are highlighted in bold.

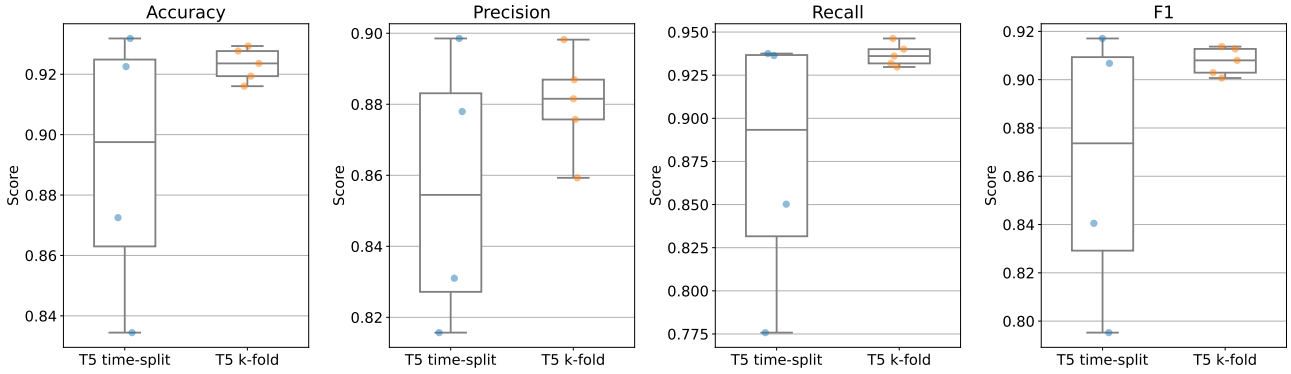

Figure 4.1: Effectiveness metrics computed over folds (box plots).

performance (except for the recall score), while k-fold has better average performance and less variance. The high score and low variance of the metrics in the k-fold set-up indicate that our model has a high generalizability.

4.3 RQ2: Comparison between T5 and baseline in time series split cross-validation setup

The second question requires to repeat the experiments with the chosen baseline and compare the metrics. Due to constraints imposed by the baseline models, certain limitations must be respected during experiments. Specifically, time-series models cannot be evaluated using k-fold cross-validation, as these models can only make inferences based on sequences seen during the training phase. It is therefore necessary to compare T5 with other models using a time series split approach. Due to a lack of data for training, baseline models are unable to handle fold 1 and will only be tested from fold 2 onwards. Moreover, models based on RNNs (LSTMs, GRUs) cannot process sequences with a length of 1, hence they need to be removed from the dataset. The hyperparameters used to construct and train the baseline models are presented concisely in Table 4.2.

Table 4.3 illustrates the maximum, mean, and minimum scores obtained in folds for each model and metric. Figure 4.2 compares models using box plots. As in Figure 4.1, the x-axes represent the models, the y-axes display the obtained scores, and the points within the box plots represent individual measurements. From the obtained results we notice that our model outperforms the baseline, having the highest maximum, average and minimum scores for all metrics.

The precision and recall box plots reveals that nearly every model demonstrates good performance in precision. However, distinctions in recall are more prominent. Notably, transformer-based models, such

	MLP	**NHiTS**	**TFT**	**GRU**	**LSTM**
Training steps	500				
Learning rate	0.001				
Stacks		3			
Blocks		1 per stack			
Layers	2	2 per block		2 (encoder) / 2 (decoder)	
Attention heads			4		
Hidden layers size	32	512	20	200 (encoder) / 200 (decoder)	
Scaler	Min-max				
Loss	Negative log-likelihood Bernoulli			Negative log-likelihood Gaussian	
Random seed	1				

Table 4.2: Hyperparameters of baseline models

Model	**Statistic**	**Accuracy**	**Precision**	**Recall**	**F1**
GRU	max	0.804	0.798	0.687	0.738
LSTM	max	0.793	0.798	0.650	0.716
MLP	max	0.806	0.873	0.621	0.716
NHITS	max	0.845	0.864	0.788	0.791
T5	max	**0.932**	**0.899**	**0.938**	**0.917**
TFT	max	0.883	0.852	0.864	0.856
GRU	mean	0.762	0.726	0.648	0.684
LSTM	mean	0.758	0.753	0.591	0.660
MLP	mean	0.775	0.794	0.598	0.681
NHITS	mean	0.805	0.780	0.713	0.744
T5	mean	**0.890**	**0.856**	**0.875**	**0.865**
TFT	mean	0.805	0.754	0.831	0.781
GRU	min	0.731	0.688	0.570	0.625
LSTM	min	0.725	0.672	0.533	0.628
MLP	min	0.727	0.687	0.567	0.621
NHITS	min	0.738	0.686	0.622	0.653
T5	min	**0.834**	**0.816**	**0.776**	**0.795**
TFT	min	0.679	0.561	0.771	0.678

Table 4.3: Average, maximum and minimum effectiveness metrics computed over time series split folds for all baseline. Cfr Table 4.1

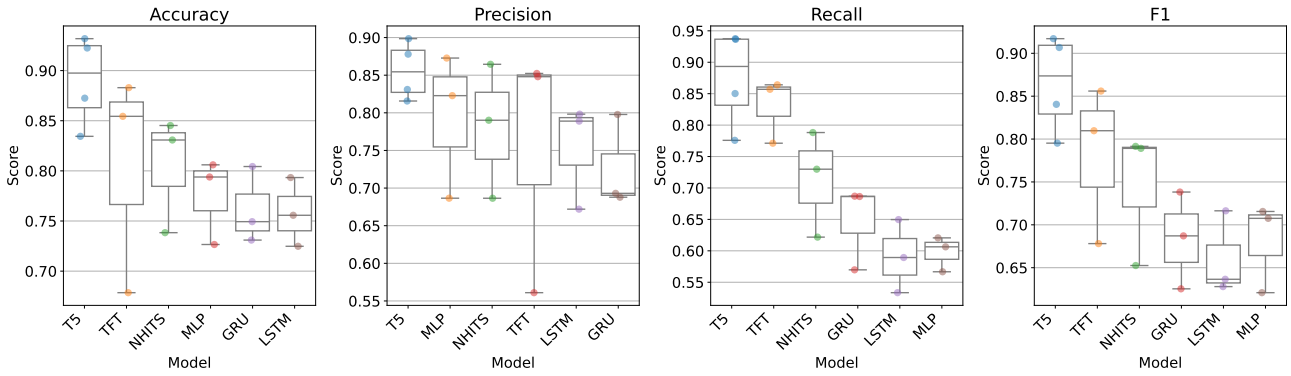

Figure 4.2: Effectiveness metrics computed over folds for all models. Box plots are ordered by maximum value.

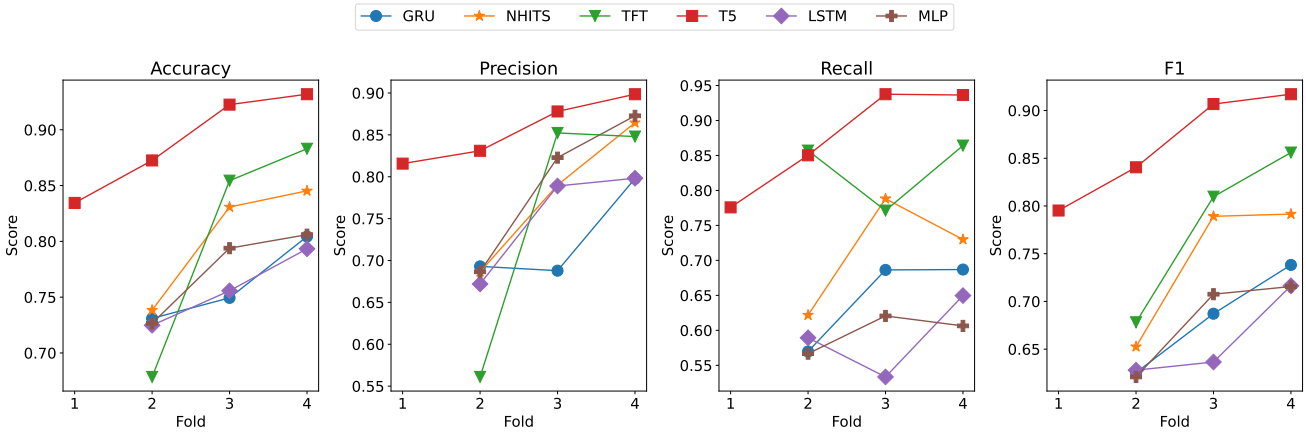

Figure 4.3: Trends of effectiveness metrics computer over folds for all models.

as T5 and TFT, consistently outperform RNN-based models (LSTM and GRU), even in their minimum values. This observation suggests that the models exhibit a high degree of reliability, implying that if a challenge is recommended, it is highly probable that the user will complete it. Nevertheless, the low recall scores indicates that some models tend to underestimate user capabilities.

4.4 RQ3: Comparison among fold between T5 and baseline

Figure 4.3 shows the metric scores obtained for each fold of the time series splitting cross-validation. The x-axes represent the folds of the time-split setup, while the y-axes contain the metrics scores. Every model is identified by color and marker shape. This plot highlights that our models outperform the others in every fold for all metrics, except for recall at fold 2 where TFT performs slightly better. Additionally, there is a growing trend of all metrics in almost all models. This demonstrates that historical data is an important predictor of challenge outcomes.

When considering the T5 metrics in Figure 4.3, the scores show almost linear growth between fold 1 and fold 3, but slow down in the last fold. The only exception to this trend is precision, where the slope of the plot remains almost constant across folds. These observations suggest that as the sequence length increases, there will likely be a point where the growth becomes negligible. However, due to the T5 input size limit, we are unable to verify this hypothesis with the current setup.

4.5 Inference analysis

In this section we analyze the prediction obtained by our model during the last fold of the time-splitting cross-validation, which is the one with the best performance and comes closest to a real use of the model. We aim to understand how the model behaves in predicting the outcome of a challenge and if there are any patterns or biases in the predictions. Note that the way the folding was constructed (see section 3.4), the samples to be predicted corresponded to all the challenges in the dataset.

The main tool for analyzing the predictions is the confusion matrix, which is a table that summarizes the predictions made by a classifier. The rows of the matrix represent the actual class of the data, while the columns represent the predicted class. The diagonal of the matrix contains the correct predictions, while the off-diagonal elements contain the incorrect predictions. For example, the value of the first row and second column in matrix (A) of Figure 4.4 indicates that the model predicted 242 samples as 'COMPLETED', while they are originally 'FAILED.

Figure 4.4 displays four versions of the confusion matrix generated by our model. Matrix (A) represents the standard confusion matrix, reporting frequencies in absolute values. Matrix (B) represents the normalized confusion matrix, reporting frequencies in relative values. The matrix (C) is the normalized confusion matrix where each row is normalized, ensuring that the sum of values in each row is 100%. The matrix (D) is also a normalized confusion matrix, but with normalization applied to the columns. The normalized confusion matrix (C) is useful for understanding how the model handles different classes and any potential imbalances, as we expect each class to have a similar percentage of correct predictions. The normalized confusion matrix (D) is a useful tool for understanding how the model handles different predictions. It is important to ensure that every prediction has a similar percentage of correct predictions. The matrix (C) shows that the model correctly handled the class imbalance in the dataset, as the percentages of correctly guessed samples are similar for both classes and are very high. The matrix (D) shows some differences in the model's performance between the 'FAILED' and 'COMPLETED' classes. Although the guessed percentage remains above 90% for both classes, the model performs better in predicting the 'FAILED' class. However, it tends to overestimate users' performance, resulting in almost 10% incorrect predictions for the 'COMPLETED' class. This behavior is acceptable for our purposes. In fact, to build a ranking of challenges and recommend the best, it is better to have a larger set of suitable challenges to choose from. This reduces the risk of not having enough challenges to recommend or having a limited diversity between them.

Figure 4.5 displays the metrics obtained by grouping the dataset according to various features and calculating the four metrics: accuracy, precision, recall and F1-score for each group. The bar graph (E) represents the metrics grouped by `counterName`, i.e. the type of challenge, while the bar graph (M) does the same, but with respect to the duration of the challenge. No critical issues are reported from these plots. In all cases, the metrics exceed a score of 85%, with some peaks approaching a perfect score. It is also worth noting that the recall is consistently higher than the precision. These two metrics refer to the 'COMPLETED' class, confirming the hypothesis made by analyzing the confusion matrices: the model has a slight tendency to overestimate user capabilities and predict more challenges as 'COMPLETED' than are actually completed. In Section 3.3, we analyzed the data set and found that our users prefer bicycles as their favorite vehicle. The model also seems to have recognised this trend, resulting in a worse

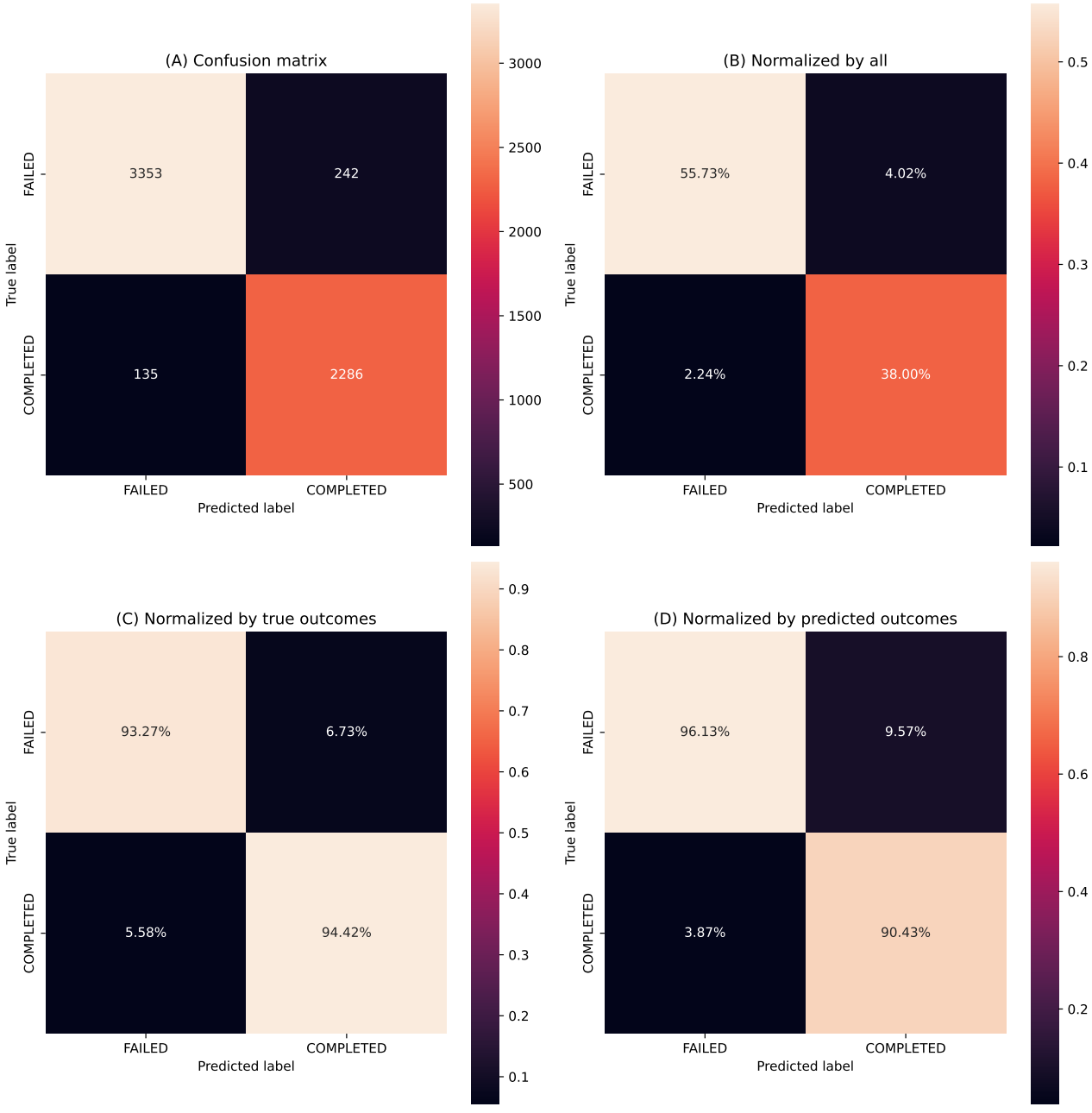

Figure 4.4: Confusion matrices of T5 computed at fold 4 (time-split)

accuracy for this type of challenge than other metrics. However, the lower recall rate for bike challenges compared to walk challenges suggests that the model may have misclassified some 'COMPLETED' bike challenges due to their underrepresentation in the dataset. On the contrary, all metrics for walk challenges are well above 90%, with a recall rate of 99%. Taking into account the distribution of the outcomes for this challenge, the results are noteworthy. See Figure 3.8 and the corresponding histogram (G) for more information. When observing green leaves challenges, it is noticeable that the metrics overlap with the 'more days in a row' columns in the barplot (F). This is because these categories overlap in the data set. Specifically, a green leaves challenge is almost always a 'more days in a row' challenge, and 'more days in a row' challenges are always green leaves challenges. For these challenges, the metrics are slightly worse than the others. However, they are a particular type of challenge that probably requires ad-hoc analysis.

In violinplot (G), we computed metrics by aggregating predictions for each user in the dataset. The objective is to determine if the model maintained consistent performance for all users, considering that the `playerId` was not a feature included in the training data. In terms of precision, as the values fall within a limited range of almost 70% to 100%, this hypothesis can be confirmed. It is worth noting the triangular shape of the kernel density estimation, indicating that the model maintained a very high level of accuracy for most users, with only a few having lower values. For other metrics, the situation is more nuanced. It is observable that the graphs have an hourglass shape, indicating that, while the model correctly predicted the success of challenges for the majority of users, there is a significant fraction of users for whom the model performed poorly. These users are likely to be completely inactive during the entire campaign, resulting in them failing all assigned challenges. A 0% score in these metrics indicates that the model could have behaved in two possible ways. The first hypothesis is that the model predicted all samples as 'FAILED', correctly guessing all samples. The second hypothesis is that the model predicted all samples as 'COMPLETED', incorrectly predicting all of them. In both cases, the True Positive and False Positive components of the metrics formulas become zero, as explained in Section 3.6, rendering the entire metrics meaningless. The second hypothesis can be rejected based on the accuracy plot. If the model had completely mispredicted a user, there would be at least one user with 0% accuracy, which is not the case here.

The H plot illustrates the trend of metrics over weeks. No critical issues are reported with respect to the metric values. The graph can be divided into three parts. The first part, which spans from April 17th to May 15th, shows the metrics starting with a perfect score and then dropping in the following two weeks. During the week of 1 May, a negative peak was observed where the recall metric was higher than the precision metric by 15%. The second phase spans from May 15th to July 10th. During this period, all metrics remained stable with values consistently greater than 90%. The following phase, from July 10 to August 28, was characterized by high variability in metric values and significant differences between them. Two negative peaks were reported during this phase. On 17 July, the first metric drop occurred, particularly in precision, compared to the previous week. However, it immediately increased during the following week. Another notable week was August 8th, where a situation similar to May 1st was observed. In this case, the recall was higher than the precision by almost 20%. After this peak, the metrics continued to rise until the end, reaching a level similar to that of the first week. When comparing this plot with the in-week distribution of the challenges, as shown in Figure 3.8, it can be

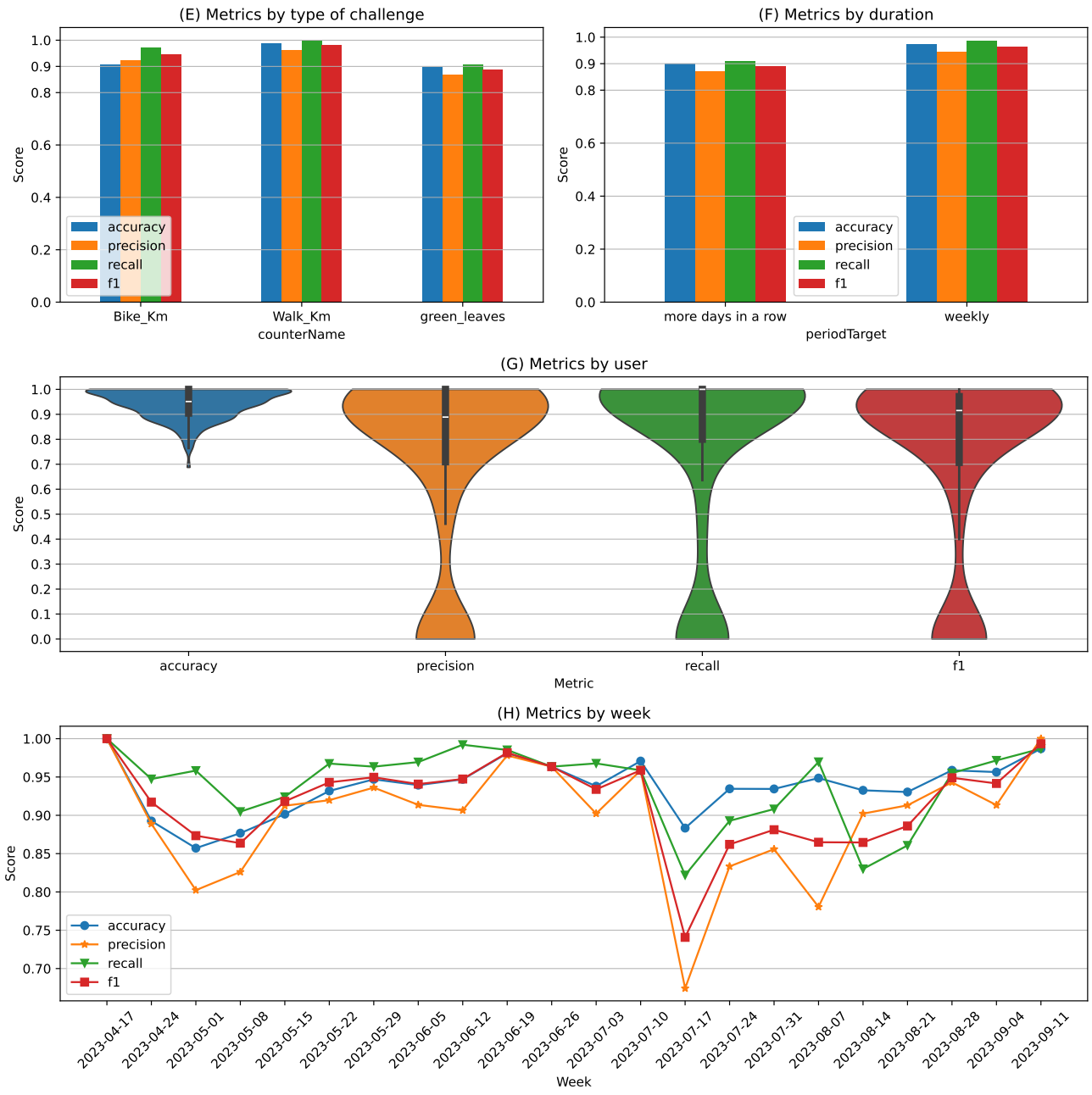

Figure 4.5: Metrics computed at fold 4 (time-split) grouping dataset by various features. Barplots (E) and (F) represent metrics grouped by type and duration of the candidate challenge. Violinplots in the second row represent the metrics calculated for each user in the dataset. Violinplots (G) represents the accuracy, precision, recall and F1-Score calculated for each user. Plot (H) are the metrics calculated by grouping by week.

observed that the periods in which there is a high variability in metrics correspond to the periods in which the completed/failed ratio is more unbalanced. Conversely, the periods in which this ratio is more balanced correspond to periods in which metrics are more stable. The negative peaks on 10 July and 8 August correspond to the weeks where the unbalancing is most evident. One possible explanation for this behavior is that during these weeks, we observed changes in user behavior that the model did not recognise well. However, after each negative peak, we noticed an immediate increase in the metrics to levels similar to those before the peaks, indicating that the model adapted quickly to the changing contexts.

5

Discussion and conclusion

The aim of this chapter is to discuss and draw conclusions from the results presented in the previous chapters. Section 5.1 recaps these results, while Section 5.2 discusses the limitations identified in this work. Additionally, Section 5.3 will describe potential future works to improve the model or integrate it into the Gamification Engine.

5.1 Recap

This thesis describes the successful fine-tuning of a pre-trained Language Model T5 to predict the outcome of a candidate challenge for a user based on their previous assigned challenge. Real data collected by the Play&Go platform during 2023 within the scope of the Ferrara Air Break project was converted into sequences of events, which were then transformed into text prompts to feed the model. The textual response of the model could be either 'COMPLETED' or 'FAILED'.

The model was fine-tuned and validated using two different cross-validation approaches: k-fold and time series split. The former verified the model's ability to handle unseen sequences, while the latter assessed its ability to continue known sequences and compared it with other deep learning models for sequential data. To assess the performance, we used four characteristics of the classification problems in their binary version: accuracy, precision, recall, and F1 score. The precision, recall and F1 score were calculated with respect to the 'COMPLETED' class.

In both setups, the results are promising. The k-fold setup showed low variance in all metrics, indicating a high generalisability of the model. The time series split setup confirmed that our approach is competitive with state-of-the-art deep learning models and highlighted the importance of historical data in our predictions.

It should be noted that all the results were obtained from a relatively small data set. This suggests that an update of the Gamification Engine based on our model could be initiated using the existing data without the need for additional data collection. In general, satisfactory performance can be achieved without requiring a massive amount of data. This allows for a relatively fast training phase, even with non-high-end hardware. In our work, the model was trained in approximately 40 minutes using a GPU Nvidia RTX 3090.

The main advantage of our text-based approach compared to classical deep learning approach is the

flexibility of the input. Leaving aside the limitations on the length of the input, by representing inputs in textual form, we can easily add descriptive features that would be difficult to represent accurately with a numerical-vector approach. One such feature in this case may be the user's profile, including their location and habits, which can easily be expressed in textual form.

The model's ability to maintain anonymity was a significant achievement, as it achieved impressive performance without including any personal information that could identify the user in the training data. This aspect is essential in terms of protecting user privacy.

5.2 Limitations

The work presented in this thesis is limited in several ways. The most important limitation lies in the fact that the model is not an out-of-the-box RS, but it is just the model that predict user/item ratings (see schema in Figure 1.1). The model was evaluated solely based on its prediction accuracy, without considering other crucial aspects of RSs such as diversity, coverage, or serendipity. These metrics heavily rely on how the predictions are utilized to create recommendations. Therefore, an end-to-end evaluation is necessary to assess these metrics. In this case, to build a complete RS, a generator or a repository for the candidate challenges and a method to rank the candidates is required. These are important points of discussion in Section 5.3.

Section 4.4 emphasized the significance of historical data for our purposes. However, the maximum number of input tokens of the model limits us. In this thesis, we used a coarse prompting schema that contained redundant or missing information. For instance, challenges assigned during the same week were ordered arbitrarily instead of contemporaneously. Moreover, both contain identical information about the user's trips since they refer to the same week. To optimize token consumption, a different prompting schema should be tested. Another possible approach to handle this problem could be to try models which accept more tokens in input than T5, such as Llama [Touvron et al., 2023] or GPT [Radford et al., 2019].

Another limitation of our study is the use of a limited dataset. Although we noted in Section 5.1 that the model can learn from a small number of data samples, it is important to acknowledge that the data we used were collected and extracted from a single city and within a specific context where all users were encouraged to achieve the same mobility goals, which may introduce bias. For instance, the Bike2Work campaign aimed to increase bicycle usage.

5.3 Future works

This study examines the feasibility of using a Large Language Model to build a Gamification Engine. However, the model cannot directly assign challenges to users as it is only a binary predictor for a given candidate. Therefore, a method is needed to propose candidates to the model and to choose between feasible ones. This second aspect is particularly critical because the effectiveness of the final system is directly correlated with it, and the model apparently does not provide any information to rank challenges, such as a probability value. Moreover, from a gamification perspective, a method to estimate difficulty and a reward calculator are missing.

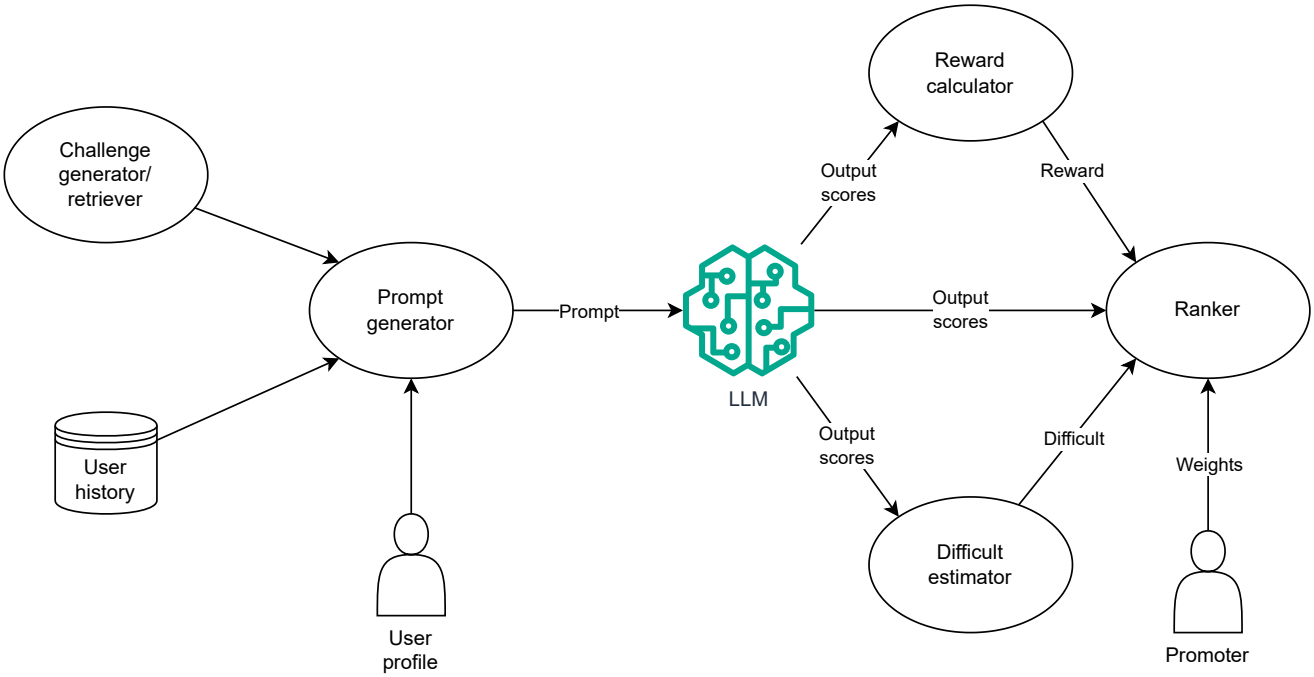

Figure 5.1: An architecture proposal for an LLM-based Gamification Engine

To integrate the model in Play&Go, we propose two possible approaches. The first and the simplest consist in substituting the current ML Module with our LLM, keeping the rest of the system almost unchanged. In this case the model is used just for its capability of outperforming the current model [Khoshkangini et al., 2017] and to get more accurate predictions for the F&S module.

The second proposal, instead, is similar to that proposed by Yue et al. [2023] in their LlamaRec framework, which ranks candidates using the scores of the tokens retrieving them from the LLM Head (see Appendix D). These scores can be seen as an indirect measure of the certainty of the model: if the model assigns a higher value to a token than to the other tokens, this means that the model considers this token to be important for the output sequence. These scores can be normalized into probability values by applying a softmax function to them, or aggregated to obtain a scalar value that can be used to rank candidates pointwise, or to estimate difficulty and calculate rewards.

Figure 5.1 shows an architecture proposal for a Gamification Engine based on an LLM. The engine is composed of six components: the challenge generator, the prompt generator, the LLM, a difficult estimator, a reward calculator and a ranker. The challenge generator generates or retrieves a set of candidate challenges to recommend. The challenge could be generated using the algorithms already implemented or a Generative AI model, such as ChatGPT. The prompt generator, as the name suggests, combines the candidates with the user's historical data with its profile and other contextual information, to get the textual prompts to be passed to the LLM. The LLM is the "core" of the system. It takes into account the prompts to get the textual response and the output tokens' scores. These information will be passed as input for the reward calculator, which calculates the prize based on the scores, the difficult estimator, which estimates the difficulty with the same modalities, and the ranker. The ranker chooses the challenges to assign to the user leveraging the information coming from previous components and heuristics, like promoters' objective or constraints on the challenges' diversity.

To implement the ranker various approaches can be tried, from the "classical" weighted average of

the inputs to more complex Learning to Rank approaches. For Learning to Rank we intend a set of supervised ML methods trained to sort lists of items for their relevance with respect to a given query, in this case the input information. It is a classical task of Web Search Engine. Examples of Learning to Rank methods include LambdaMART and its variants [Burges, 2010], but more recently also LLM are used for this purpose with promising results [Qin et al., 2023].

III

Appendices

A

Dataset Preprocessing

In this appendix we will see how we converted the raw dataset (as described in Section 3.3) firstly into sequences, then into text prompts.

A.1 From data to sequences

```
import pandas as pd

def gen_windows(x, sequence_length):
    if len(x) < sequence_length:
        return [x[:i] for i in range(1, len(x)+1)]
    else:
        return
            [x[:i] for i in range(1, sequence_length+1)] +
            [x[i:i+sequence_length] for i in range(1, len(x)-sequence_length+1)]

dataset = df.groupby('playerId').agg(list).reset_index()
for col in df.columns[1:]:
    dataset[col] = dataset[col].apply(lambda x: gen_windows(x, 5))

dataset = dataset.explode(df.columns[1:].to_list()).reset_index(drop=True)
dataset.drop(columns=['playerId'], inplace=True)
dataset['unique_id'] = dataset.index
dataset = dataset.explode(dataset.columns[:-1].to_list()).reset_index(drop=True)
dataset['ds'] = dataset.groupby('unique_id').cumcount()
```

First, we group the records by player, then we aggregate them all to create the user's history as a Python list. From this list we create the "windows" as shown in figure 3.2 by applying `gen_windows` to all columns (except the first, the `playerId` column). After this operation we have a row for each player where each column is a list of windows. So we apply the `explode` method, which transforms each element of a list-like list into a row, replicating the index values. In this way, we obtain a row for each

window, identifying each of them by its index. Finally, we apply a second `explode` and we insert the id in sequence with the `cumcount` method. In this way, we obtained a dataset in which each row is a single challenge, but with the columns `unique_id` and `ds` identifying it in its sequence. These column names are not arbitrary (see Appendix C).

```
23 def serialize(x):
24     return {
25         'City': x['TerritoryId'],
26         'Challenge': challenge_to_text(x['counterName'], x['target'], x['periodTarget']),
27         'Trips': trips_to_text(x['bike'], x['bus'], x['car'], x['train'], x['walk']),
28     }
29 
30 dataset['challenge_data'] = dataset.apply(serialize, axis=1)
```

This snippet of code takes the information about the challenge in question and inserts it into a Python dictionary.

A.2 From sequences to prompts

```
34 def make_query(x):
35     for d, s in zip(x['challenge_data'], x['state']):
36         # Associate each challenge with its outcome
37         d['Outcome'] = s
38     return str(x['challenge_data'])
39 
40 def prompts_dataset(dataset):
41     prompts = dataset.groupby('unique_id')[['challenge_data', 'state']].agg(
42         list).reset_index(drop=True)
43     prompts['y'] = prompts['state'].apply(lambda x: x[-1]) # Ground thruth
44     prompts['state'] = prompts['state'].apply(lambda x: x[:-1] + ['?'])
45     prompts['X'] = prompts.apply(make_query, axis=1)
46     return prompts[['X', 'y']] # Prompts with their desired output
```

To convert sequences into prompts, firstly we regroup them into lists. The ground truth is obtained from the outcome of the last challenge (row 42) and substituted with a '?'.

A.3 Cross validation

```
48 # Time series split cross-validation
49 for fold in range (2, 5):
50     test_df = dataset[dataset['ds'] <= fold]
51     # Remove the last challenge for the train dataset
52     train_df = test_df.drop(test_df.groupby('unique_id').tail(1).index)
```

```
53      test, train = prompts_dataset(test_df), prompts_dataset(train_df)
54      ... # Model train and test
55
56  # K-fold cross-validation
57  from sklearn.model_selection import StratifiedKFold
58
59  ds = prompts_dataset(dataset)
60  skf = StratifiedKFold(n_splits=5, shuffle=True)
61  for train_index, test_index in skf.split(ds['X'], ds['y']):
62      train, test = ds.iloc[train_index], ds.iloc[test_index]
63      ... # Model train and test
```

The code from row 49 to row 54 implements the time series split cross-validation. The test dataframe is obtained filtering the raw dataset using the `ds` column. The train data frame is obtained from the test dataset removing the last challenge from every sequence. These two data frames are finally converted into prompt datasets.

The rows from 57 to 63 implement the k-fold cross validation using the `scikit-learn` library[1]. In this case we used the stratified k-fold schema. This schema is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set.

[1] `https://scikit-learn.org/stable/index.html`

B

HuggingFace Transformers

In this appendix we will focus on how to use the Transformers python library for training a T5 model to generate text.

Transformers[1] is a Python library developed by the HuggingFace team[2]. This library provides APIs and tools to train and use Tranformer-based pretrained models, including T5, BART[3], BERT[4], GPT[5], Llama[6], Mistral[7] and their variants, like RoBERTa, an optimized version of BERT, and ByT5, a T5 variant that tokenizes input byte-per-byte instead of using words or subwords units. These models support NLP tasks like text classification, language modeling and text generation, but there are models that can handle multimodal inputs like video or audio.

Transformers is builded as an abstraction of other frameworks used for implementing deep learning models like PyTorch[8], TensorFlow and JAX.

B.1 Model loading

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained('t5-base', model_max_length=512)
model = AutoModelForSeq2SeqLM.from_pretrained('t5-base')
```

Firstly, we have to load the pre-trained model and its tokenizer from the Huggingface Hub. The `tokenizer` is a Python object that converts the text into token sequences based on the vocabulary of the model and vice versa. The `model` object is the Pytorch module that implements the desired model. The `from_pretrained` method also initializes the model's parameters with the ones of the pretrained model.

[1] https://huggingface.co/docs/transformers/index
[2] https://huggingface.co/huggingface
[3] https://huggingface.co/docs/transformers/model_doc/bart
[4] https://huggingface.co/docs/transformers/model_doc/bert
[5] https://huggingface.co/docs/transformers/model_doc/openai-gpt
[6] https://huggingface.co/docs/transformers/model_doc/llama2
[7] https://huggingface.co/docs/transformers/model_doc/mistral
[8] https://pytorch.org/

B.2 Input preprocessing

```
8  from datasets import Dataset
9
10 def preprocess(examples):
11     inputs = [doc for doc in examples["text"]]
12     model_inputs = tokenizer(inputs, max_length=512, truncation=True)
13     labels = tokenizer(text_target=examples["target"], max_length=32, truncation=True)
14     model_inputs["labels"] = labels["input_ids"]
15     return model_inputs
16
17 train_ds = Dataset.from_dict({'text': X, 'target': y}).map(preprocess, batched=True)
```

This snippet of code preprocesses the training dataset, previously divided in input prompts `X` and target texts `y`, converting them into sequence of tokens for the model. The parameter `truncation` in rows 12 and 13 means that if a text exceeds the number of token indicated in `max_length` parameter, the sequence has to be truncated. The code in line 14 is necessary for the Transformers library to recognize the ground-truth for the training.

This preprocessing phase is done encapsulating the raw data into a `Dataset` object, and the `map` method divides it into batches and calls the `preprocess` function in parallel for each batch.

B.3 Training

```
21 from transformers import DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer
22
23 data_collator = DataCollatorForSeq2Seq(tokenizer, model='t5-base')
24
25 training_args = Seq2SeqTrainingArguments(
26     output_dir=CACHE_DIR,
27     evaluation_strategy="no",
28     save_strategy="no",
29     num_train_epochs=3,
30     predict_with_generate=True,
31     no_cuda=False,
32     per_device_train_batch_size=2,
33     optim="adamw_torch",
34 )
35
36 trainer = Seq2SeqTrainer(
37     model=model,
38     args=training_args
39     train_dataset=train_ds,
```

```
40      data_collator=data_collator
41  )
42
43  trainer.train()
```

The code snippet above executes the code to fine-tune the model. The `data_collator` is the object responsible to form the training batches for the model, converting the tokens into tensors and shuffling the samples for the Stochastic Gradient Descent algorithm. Variables `trainer` and `training_args` encapsulate the training algorithm and its hyperparameters[9], such as the number of epochs or the size of training batches.

The training phase is executed by the `train` method.

B.4 Text generation

```
47  import torch
48
49  def batch_predicts(examples: Dataset):
50      inputs = [doc for doc in examples["text"]]
51      model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding=True)
52      input_ids = torch.LongTensor(model_inputs["input_ids"]).to(model.device)
53      attention_mask = torch.LongTensor(model_inputs["attention_mask"]).to(model.device)
54      outputs = model.generate(
55          input_ids=input_ids,
56          attention_mask=attention_mask,
57          max_length=32,
58          do_sample=False
59      )
60      output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)
61      return {"y_hat": output_str}
62
63  ds = Dataset.from_dict({'text': X_hat})
64  ds = ds.map(batch_predicts, batched=True, batch_size=32)
65  ds['y_hat'] # Get generated text
```

This code takes a vector of texts `X_hat` and for each of them returns the output from the fine-tuned `model`, using a schema similar to that seen for the preprocessing, with some little differences. In row 50 the parameter `padding` indicates that prompts shorter than 512 token are padded using a special token. The final output is obtained using the `generate` method, which takes the tokens sequence and the attention mask[10] to generate the output sequence of tokens. In this case, the input token sequence is represented by the indexes of the tokens in the model's vocabulary, rather than the actual textual tokens

[9] `https://huggingface.co/docs/transformers/v4.37.2/en/main_classes/trainer#transformers.Seq2SeqTrainingArguments`

[10] The attention mask is a binary vector indicating the position of the padded indices so that the model ignore them

themselves. This sequence is converted into text with the `batch_decode` method of the `tokenizer`. The `do_sample=False` parameter activates the greedy generation strategy.

C

Baseline with Neural Forecast

In this appendix we see the code for implementing the baseline (see Section 3.2) using the NeuralForecast framework. Also NeuralForecast is implemented as an abstraction layer of PyTorch.

C.1 Preprocessing

```
import pandas as pd

dataset.rename(columns={'state': 'y'}, inplace=True)
dataset['y'] = dataset['y'].map({'FAILED': 0, 'COMPLETED': 1})
dataset = pd.get_dummies(dataset, columns=['counterName', 'TerritoryId'], dtype=int)
```

NeuralForecast uses Pandas dataframes as inputs for the models. In particular, the dataframe must include the following columns: `unique_id`, `ds` and `y`, where `unique_id` identifies individual time series from the dataset, `ds` is the date, and `y` is the target variable. The `y` columns must not have missing or non-numeric values.

Since NeuralForecast models accept only numeric vectors as input, we have to convert categorical columns with one-hot encoding and binary columns with 0/1 mapping.

C.2 Models initialization

```
from neuralforecast import NeuralForecast
from neuralforecast.losses.pytorch import DistributionLoss
from neuralforecast.models import TFT, MLP, NHITS

futr_exog_list = [
    'target', 'periodTarget', 'counterName_Bike_Km', 'counterName_Walk_Km',
    'counterName_green_leaves', 'TerritoryId_Ferrara',
    'bike', 'bus', 'car', 'train', 'walk'
]
```

```
hist_exog_list = []
stat_exog_list = []
```

Firstly, we have to define the exogeneous variables. Exogenous variables are predictors that are independent of the model being used for forecasting and provide additional information to improve forecasting accuracy. We classify the exogenous variables in three categories by whether they reflect static or time-dependent aspects of the data.

The **static exogenous variables** carry time-invariant information for each time series. Examples of static variables include designators such as identifiers of regions, groups of products, *etc.* **Historic exogenous variables** are time-dependent variables restricted to past observed values, so they are *not known* at the time of the prediction. Their predictive power depends on how past values can provide significant information about future values of the target variable. **Future exogenous variables** are the values available at the time of the prediction. In our case we can consider all columns different from `unique_id`, `ds` and `y` as future exogenous variables.

```
args = {
    'h': 1,
    'input_size': 5,
    'loss': DistributionLoss('Bernoulli'),
    'max_steps': 500,
    'scaler_type': 'minmax',
    'val_check_steps': 250,
    'futr_exog_list': futr_exog_list,
    'hist_exog_list': hist_exog_list,
    'stat_exog_list': stat_exog_list,
    'start_padding_enabled': True   # Pad sequences if shorter than input_size
}

models = [
    MLP(
        **args,
        hidden_size=32,
    ),
    NHITS(
        **args,
    ),
    TFT(
        **args,
        hidden_size=20,
        windows_batch_size=None
    )]
```

Then we define the hyperparameters (see Table 4.2). The horizon `h` indicates how many values the

models have to predict from the training data, in this case we need just one value. The `input_size` is the maximum length of the sequences in input. The `loss` parameter indicates the loss function to be used for the training. The `max_steps` are the maximum number of training steps, while `val_check_steps` indicates every how many training steps to launch a validation phase. The code below set hyperparameter only for MLP, NHiTS and TFT model, for the other two models (GRU and LSTM), we have to introduce some differences.

```
50 from neuralforecast.models import GRU, LSTM
51
52 args = {
53     'h': 1,
54     'input_size': 5,
55     'loss': DistributionLoss('Normal'),
56     'max_steps': 500,
57     'scaler_type': 'minmax',
58     'val_check_steps': 250,
59     'futr_exog_list': futr_exog_list,
60     'hist_exog_list': hist_exog_list,
61     'stat_exog_list': stat_exog_list
62 }
63
64 models = [
65     GRU(
66         **args,
67     ),
68     LSTM(
69         **args,
70     )
71 ]
72
73 # Drop sequences shorter than 1
74 dataset = dataset.groupby('unique_id').filter(lambda x: len(x) > 1)
```

Respect the other models, the implementation of RNN-based models in NeuralForecast doesn't support the Bernoulli distribution as loss function[1] and they cannot accept sequence of length less than 2, so we have to fall back to the Normal distribution negative log-likehood as loss function and drop sequences shorter than 1.

[1] https://nixtlaverse.nixtla.io/neuralforecast/examples/temporal_classifiers.html#fit-the-models

C.3 Cross-validation

```
78 id2label = {1: 'COMPLETED', 0: 'FAILED'}
79
80 for i in range(2, 5):
81     train_df = dataset[dataset['ds'] <= i]
82
83     # the freq parameter is required but it is meaningless for us
84     nf = NeuralForecast(models=models, freq='D')
85
86     # Train and test all models
87     Y_hat = nf.cross_validation(train_df, n_windows=1, use_init_models=True)
88     Y_hat['y'] = Y_hat['y'].map(id2label)
```

For time split cross validation follow the same schema seen in Section A.3. In this case the training and the testing phase are encapsulated in the `cross_validation` method of NeuralForecast. Setting `n_windows=1` mirrors a traditional train-test split in which the model are trained using sequences until $len(x) - 1$, then as test it has to predict the last observation of each sequence.

```
91
92 # Get puntual predictions for MLP, NHITS and TFT
93 Y_hat['MLP'] = Y_hat['MLP'].apply(lambda x: 1 if x > 0.5 else 0).astype(int).map(id2label)
94
95 # Get puntual predictions for GRU and LSTM
96 mean = Y_hat['GRU'].mean()
97 Y_hat['GRU'] = Y_hat['GRU'].apply(lambda x: 1 if x > mean else 0).astype(int).map(id2label)
```

The predictions from the models are returned from the `cross_validation` method as a Pandas dataframe as floating-point values, so to convert them into puntual predictions we simply set a threshold value and converted the values into 0 or 1. For models that used Bernoulli distribution the predictions are expressed as values in range $[0, 1]$ so the threshold is set to 0.5, while for other models the predictions are real numbers, so the thredshold is set on the average value.

D

Text Generation Utils and Output Scores Retrieval with HF Transformers

In this Jupyter Notebook we will see an example of a use of the model for making recommendations. In particular, after defining some prompts with different candidate challenges, we will extract the output token sequence with their normalized scores.

Firstly, we have to load the fine-tuned model and the tokenizer.

[1]:
```
model_name = '../../models/t5-base/'
# Load model directly
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

Then, we create three prompts with three possible challenge to assign: - collect 30 leaves for 2 days in a row, - bike for 1 km, - walk for 2 km

[2]:
```
promptA = "[{'City': 'Ferrara', 'Challenge': 'walk for 1 km', 'Trips':␣
 ↪{'bike': 0.0, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.0},␣
 ↪'Outcome': 'FAILED'}, {'City': 'Ferrara', 'Challenge': 'collect 30␣
 ↪leaves for 2 days in a row', 'Trips': {'bike': 0.05, 'bus': 0.0, 'car':␣
 ↪0.0, 'train': 0.0, 'walk': 0.0}, 'Outcome': '?'}]"
promptB = "[{'City': 'Ferrara', 'Challenge': 'walk for 1 km', 'Trips':␣
 ↪{'bike': 0.0, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.0},␣
 ↪'Outcome': 'FAILED'}, {'City': 'Ferrara', 'Challenge': 'bike for 1 km',␣
 ↪'Trips': {'bike': 0.05, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.
 ↪0}, 'Outcome': '?'}]"
promptC = "[{'City': 'Ferrara', 'Challenge': 'walk for 1 km', 'Trips':␣
 ↪{'bike': 0.0, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.0},␣
 ↪'Outcome': 'FAILED'}, {'City': 'Ferrara', 'Challenge': 'walk for 2 km',␣
 ↪'Trips': {'bike': 0.05, 'bus': 0.0, 'car': 0.0, 'train': 0.0, 'walk': 0.
 ↪0}, 'Outcome': '?'}]"
```

Now, we feed the prompts to the model. We set the `output_scores` and `return_dict_in_generate` equal to `True` to get the scores from the model head. We obtain a tensor for each token of the final sequence, each of dimension $1 \times |V|$. Then, we convert the scores into probability vectors using the softmax function. The output text is obtained concatenating the tokens with highest scores.

[3]:
```
import numpy as np
from scipy.special import softmax

def get_scores(text):
    inputs = tokenizer(text, return_tensors="pt").input_ids
    outputs = model.generate(inputs, max_new_tokens=32, do_sample=False,␣
 ↪output_scores=True, return_dict_in_generate=True)
    scores = np.stack([softmax(x[0]) for x in outputs.scores])
```

```
    return [{
        'id': id,
        'token': tokenizer.decode([id], skip_special_tokens=True),
        'score': score
    } for id, score in zip(np.argmax(scores, axis=1), np.max(scores,␣
↪axis=1))][:-1] # Remove the EOS token
```

[4]:
```
get_scores(promptA)
```

[4]:
```
[{'id': 23666, 'token': 'COMP', 'score': 0.919757},
 {'id': 3765, 'token': 'LE', 'score': 1.0},
 {'id': 11430, 'token': 'TED', 'score': 1.0}]
```

[5]:
```
get_scores(promptB)
```

[5]:
```
[{'id': 23666, 'token': 'COMP', 'score': 0.9967939},
 {'id': 3765, 'token': 'LE', 'score': 1.0},
 {'id': 11430, 'token': 'TED', 'score': 1.0}]
```

[6]:
```
get_scores(promptC)
```

[6]:
```
[{'id': 377, 'token': 'F', 'score': 0.99999917},
 {'id': 22862, 'token': 'AIL', 'score': 1.0},
 {'id': 2326, 'token': 'ED', 'score': 1.0}]
```

Analyzing the outputs of the three scores, the model predicted two challenges as "COMPLETED" and one as "FAILED". Focusing on the scores of the token, we notice that for they are different only for the firsts tokens. These scores can be used to rank the candidates.

Bibliography

A. Acharya, B. Singh, and N. Onoe. LLM Based Generation of Item-Description for Recommendation System. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1204–1207, Singapore Singapore, Sept. 2023. ACM. ISBN 9798400702419. doi: 10.1145/3604915.3610647. URL `https://dl.acm.org/doi/10.1145/3604915.3610647`.

A. Al-Molegi, M. Jabreel, and A. Martínez-Ballesté. Move, attend and predict: An attention-based neural model for people's movement prediction. *Pattern Recognition Letters*, 112:34–40, 2018.

S. Bassanelli, A. Bucchiarone, and F. Gini. Gamidoc: The importance of designing gamification in a proper way. *IEEE Transactions on Games*, pages 1–19, 2024. doi: 10.1109/TG.2024.3364061.

F. Bonetti, A. Bucchiarone, A. Cicchetti, and A. Marconi. Challenging models: Formalizing quests in gamified systems for behavioral change. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 747–756. IEEE, 2023.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

A. Bucchiarone, S. Bassanelli, M. Luca, S. Centellegher, P. Cipriano, L. Giovannini, B. Lepri, and A. Marconi. Play&go corporate: An end-to-end solution for facilitating urban cyclability. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

I. Caponetto, J. Earp, and M. Ott. Gamification and education: A literature review. In *European Conference on Games Based Learning*, volume 1, page 50. Academic Conferences International Limited, 2014.

C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski. N-hits: Neural hierarchical interpolation for time series forecasting, 2022.

J. Chang, C. Gao, Y. Zheng, Y. Hui, Y. Niu, Y. Song, D. Jin, and Y. Li. Sequential recommendation with graph neural networks. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 378–387, 2021.

V. Colizza, A. Barrat, M. Barthelemy, A.-J. Valleron, and A. Vespignani. Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions. *PLoS medicine*, 4(1):e13, 2007.

E. U. Commission. Green paper on the impact of transport on the environment. a community strategy for 'sustainable mobility', 1992.

R. Dey and F. M. Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

D. Di Palma. Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Models. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1369–1373, Singapore Singapore, Sept. 2023. ACM. ISBN 9798400702419. doi: 10.1145/3604915.3608889. URL https://dl.acm.org/doi/10.1145/3604915.3608889.

W. Fan, Z. Zhao, J. Li, Y. Liu, X. Mei, Y. Wang, Z. Wen, F. Wang, X. Zhao, J. Tang, and Q. Li. Recommender systems in the era of large language models (llms), 2023.

U. Fattore, M. Liebsch, B. Brik, and A. Ksentini. AutoMEC: LSTM-based user mobility prediction for service management in distributed MEC resources. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 155–159. ACM, 2020. ISBN 978-1-4503-8117-8. doi: 10.1145/3416010.3423246. URL https://dl.acm.org/doi/10.1145/3416010.3423246.

FBK Editorial Staff. Air Break, the Project under the European Union's Urban Innovative Actions Program is Ending. *Fondazione Bruno Kessler Magazine*, 2023. URL https://magazine.fbk.eu/en/news/air-break-the-project-under-the-european-unions-urban-innovative-actions-program-is-ending/.

S. Gambs, M.-O. Killijian, and M. N. Del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, pages 1–6. ACM, 2012. ISBN 978-1-4503-1163-2. doi: 10.1145/2181196.2181199. URL https://dl.acm.org/doi/10.1145/2181196.2181199.

Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang. Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System, Apr. 2023. URL http://arxiv.org/abs/2303.14524. arXiv:2303.14524 [cs].

C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 191–200. IEEE, 2016.

B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Y. Hou, J. Zhang, Z. Lin, H. Lu, R. Xie, J. McAuley, and W. X. Zhao. Large Language Models are Zero-Shot Rankers for Recommender Systems, May 2023. URL http://arxiv.org/abs/2305.08845. arXiv:2305.08845 [cs].

H.-P. Hsieh, C.-T. Li, and X. Gao. T-gram: A time-aware language model to predict human mobility. *Proceedings of the International AAAI Conference on Web and Social Media*, 9(1):614–617, 2021. ISSN 2334-0770, 2162-3449. doi: 10.1609/icwsm.v9i1.14663. URL https://ojs.aaai.org/index.php/ICWSM/article/view/14663.

R. Kazhamiakin, E. Loria, A. Marconi, and M. Scanagatta. A gamification platform to analyze and influence citizens' daily transportation choices. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2153–2167, 2021a. doi: 10.1109/TITS.2021.3049792.

R. Kazhamiakin, E. Loria, A. Marconi, and M. Scanagatta. A gamification platform to analyze and influence citizens' daily transportation choices. *IEEE Trans. Intell. Transp. Syst.*, 22(4):2153–2167, 2021b.

R. Khoshkangini, A. Marconi, and G. Valetto. Machine learning for personalized challenges in a gamified sustainable mobility scenario. In *Extended abstracts publication of the annual symposium on computer-human interaction in play*, pages 361–368, 2017.

R. Khoshkangini, G. Valetto, A. Marconi, and M. Pistore. Automatic generation and recommendation of personalized challenges for gamification. *User Modeling and User-Adapted Interaction*, 31:1–34, 2021.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

J. Li, Y. Wang, and J. McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 322–330, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368223. doi: 10.1145/3336191.3371786. URL https://doi.org/10.1145/3336191.3371786.

L. Li, Y. Zhang, and L. Chen. Prompt Distillation for Efficient LLM-based Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1348–1357, Birmingham United Kingdom, Oct. 2023. ACM. ISBN 9798400701245. doi: 10.1145/3583780.3615017. URL https://dl.acm.org/doi/10.1145/3583780.3615017.

B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

Z. Ma and P. Zhang. Individual mobility prediction review: Data, problem, method and application. *Multimodal Transportation*, 1(1):100002, 2022. ISSN 2772-5863. doi: https://doi.org/10.1016/j.multra.2022.100002. URL https://www.sciencedirect.com/science/article/pii/S2772586322000028.

A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. Mining user mobility features for next place prediction in location-based services. In *2012 IEEE 12th International Conference on Data Mining*, pages 1038–1043. IEEE, 2012. ISBN 978-1-4673-4649-8 978-0-7695-4905-7. doi: 10.1109/ICDM.2012.113. URL http://ieeexplore.ieee.org/document/6413812/.

B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019.

Z. Qin, R. Jagerman, K. Hui, H. Zhuang, J. Wu, J. Shen, T. Liu, J. Liu, D. Metzler, X. Wang, et al. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*, 2023.

M. Quadrana, P. Cremonesi, and D. Jannach. Sequence-aware recommender systems. *CoRR*, abs/1802.08452, 2018. URL http://arxiv.org/abs/1802.08452.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820, 2010.

F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2010.

H. Ritchie. Cars, planes, trains: where do co2 emissions from transport come from? *Our World in Data*, 2020. https://ourworldindata.org/co2-emissions-from-transport.

K. Roitero, B. Portelli, G. Serra, V. D. Mea, S. Mizzaro, G. Cerro, M. Vitelli, and M. Molinara. Detection of wastewater pollution through natural language generation with a low-cost sensing platform. *IEEE Access*, 11:50272–50284, 2023. doi: 10.1109/ACCESS.2023.3277535.

R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

S. Sanner, K. Balog, F. Radlinski, B. Wedin, and L. Dixon. Large language models are competitive near cold-start recommenders for language- and item-based preferences. In *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23, page 890–896, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702419. doi: 10.1145/3604915.3608845. URL https://doi.org/10.1145/3604915.3608845.

M. Scanagatta and A. Marconi. How to design personalized challenges for mobile motivational systems? ask your players! In *Games and Learning Alliance: 10th International Conference, GALA 2021, La Spezia, Italy, December 1–2, 2021, Proceedings 10*, pages 245–251. Springer, 2021.

F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

A. M. Toda, A. C. Klock, W. Oliveira, P. T. Palomino, L. Rodrigues, L. Shi, I. Bittencourt, I. Gasparini, S. Isotani, and A. I. Cristea. Analysing gamification elements in educational environments using an existing gamification taxonomy. *Smart Learning Environments*, 6(1):1–14, 2019.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

B. Wang and I. Kim. Short-term prediction for bike-sharing service using machine learning. *Transportation Research Procedia*, 34:171–178, 2018. ISSN 23521465. doi: 10.1016/j.trpro.2018.11.029. URL `https://linkinghub.elsevier.com/retrieve/pii/S2352146518303181`.

Y. Wang, Z. Chu, X. Ouyang, S. Wang, H. Hao, Y. Shen, J. Gu, S. Xue, J. Y. Zhang, Q. Cui, L. Li, J. Zhou, and S. Li. Enhancing recommender systems with large language model reasoning graphs, 2023.

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

M. Yan, S. Li, C. A. Chan, Y. Shen, and Y. Yu. Mobility prediction using a weighted markov model based on mobile user classification. *Sensors*, 21(5):1740, 2021. ISSN 1424-8220. doi: 10.3390/s21051740. URL `https://www.mdpi.com/1424-8220/21/5/1740`.

J. Ye, Z. Zhu, and H. Cheng. What's your next move: User activity prediction in location-based social networks. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 171–179. SIAM, 2013.

Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

Z. Yue, S. Rabhi, G. de Souza Pereira Moreira, D. Wang, and E. Oldridge. Llamarec: Two-stage recommendation using large language models for ranking, 2023.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into deep learning*. Cambridge University Press, 2023a.

C. Zhang, L. Zhang, Y. Liu, and X. Yang. Short-term prediction of bike-sharing usage considering public transport: A LSTM approach. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1564–1571. IEEE, 2018. ISBN 978-1-72810-321-1 978-1-72810-323-5. doi: 10.1109/ITSC.2018.8569726. URL `https://ieeexplore.ieee.org/document/8569726/`.

J. Zhang, R. Xie, Y. Hou, W. X. Zhao, L. Lin, and J.-R. Wen. Recommendation as Instruction Following: A Large Language Model Empowered Recommendation Approach, May 2023b. URL http://arxiv.org/abs/2305.07001. arXiv:2305.07001 [cs].

Z. Zhao, H. N. Koutsopoulos, and J. Zhao. Individual mobility prediction using transit smart card data. *Transportation Research Part C: Emerging Technologies*, 89:19–34, 2018. ISSN 0968-090X. doi: 10.1016/j.trc.2018.01.022. URL https://www.sciencedirect.com/science/article/pii/S0968090X18300676.