GlassFood: The holographic healthy smart recipe app

Bruno Moya Ruiz

Resum– Aquest projecte integra realitat mixta, visió per ordinador i aprenentatge profund per desenvolupar una aplicació per al casc Glassear, un projecte emprenedor d'avantguarda. L'objectiu de l'aplicació és millorar la salut dels usuaris i fomentar una cultura d'alimentació saludable. La realitat mixta combina la realitat virtual i la realitat augmentada, superant la bretxa entre el món real i el digital, per oferir una experiència immersiva i interactiva a l'usuari. Mitjançant l'ús de les càmeres del casc per escanejar l'entorn, l'aplicació identifica els ingredients disponibles i presenta receptes saludables amb instruccions en 3D i vista en primera persona. El component de visió per ordinador del projecte proporciona detecció d'objectes en temps real amb una alta precisió i record, més del 95%, i la interfície d'usuari avançada permet una interacció perfecta amb l'experiència, utilitzant inputs naturals com ara mans o ulls humans, fins i tot la veu. La combinació d'aquestes tecnologies dona com a resultat una experiència dinàmica i intuïtiva per a l'usuari, promovent hàbits alimentaris saludables i donant més valor a noves tecnologies.
Paraules clau– Realitat mixta, Realitat augmentada, Realitat virtual, Interacció humana amb ordinador, Interficies 3D, Visió per ordinador, Gràfics per ordinador, Yolo

Abstract– This project integrates mixed reality, computer vision, and deep learning to develop an application for the Glassear headset, a cutting-edge entrepreneurial project. The aim of the application is to improve user health and encourage a culture of healthy eating. Mixed reality combines virtual and augmented reality, bridging the gap between the real and digital world, to offer an immersive and interactive experience to the user. By using the headset's cameras to scan the environment, the application identifies available ingredients and presents healthy recipes with instructions in 3D and first-person view. The computer vision component of the project provides real-time object detection with high precision and recall, over 95%, and the advanced user interface allows for seamless interaction with the experience, by using natural input such as human hands or eyes. The combination of these technologies results in a dynamic and intuitive experience for the user, promoting healthy eating habits and giving more value to new technologies.
Keywords– Mixed reality, Augmented reality, Virtual reality, Human computer interaction, 3D interfaces, Computer vision, Computer graphics, Yolo

1 Introduction

XR or Extended Reality is a technology on the rise. Extended reality or XR is the theoretical technology that groups three existing technologies, Augmented Reality, Virtual Reality (The most known one), and Mixed reality. The last one is an aggregation of both AR and VR, providing the immersion VR gives, and the possibility to see the reality that AR gives.

Eventually, it might seem a thing of the current years, but it has been in development since the 90s. Companies like Nintendo released a red-colored graphics dizzy VR headset in 1995 called Virtual boy. It only stayed in the market for 6 months after an embarrassing exit.

- E-mail de contacte: bruno.moya@autonoma.cat
- Menció realitzada: Enginyeria de Computació
- Treball tutoritzat per: nom i cognoms del tutor (departament)
- Curs 2022/23

1.1 Project Background

8 years ago, I started a project that has evolved into a more mature and professional state. The project's philosophy is to develop a device that can enable the user to experience and enter the Extended reality technology, but having an easy and inexpensive gateway. The project has been named 'Glassear' since the first day, and it is in process of entering the Extended reality market. (Obtain financing and manufacturing).

The definition might change depending on the user knowledge or point of view, but from my perspective and conception, the Glassear [1] headset enters the category of Mixed reality, as it contains augmented reality, as you are able to see your world with your own eyes, and the immersion and sensitivity the VR Headsets do have.

As all technologies are on the rise, they need content and demonstrations in order to give them and final users more value. That is why, thinking of a project that could serve as a good thing for society and give value to Glassear and its headset, GlassFood

[1] www.glassear.com

project has been developed and explained on this article.

1.2 Project introduction

My main objective in this project is to develop software that does something good for society. All apps you may see online inside the Extended reality world are for fun, like VR games, but none of them helps improve something very important for humans, our health.

So, what I tried achieving here is a project called Glassfood that mixes the latest computer vision tech, to detect ingredients in your fridge and environment, to then recommend healthy recipes to do with those ingredients. Then, the latest graphics and algorithms of human-computer interaction so it works in a first-person and 3D view thanks to the Glassear headset, so for example, you are able to directly look at your fridge with the headset on, and 3D information appears on top of it.

This 3D information also shows the recipe steps, so you can cook while looking at them at the same time, so who knows if also Glassfood could help people start cooking. The whole article mixes neural network training with python, custom backends and scalable databases, and 3D graphics with Unity 3D.

1.3 Project Goals

The main objectives of the project are training and improving a Neural Network for computer vision and detecting the ingredients on images, then, make that neural network work inside the Unity Editor, where we will develop the Human Computer interaction and 3D interfaces for the headset. Finally, neural network is our input, so we need and output, in this case, an API where the app can find recipes to show to the user on our 3D Interface. Each step, has its different goals and milestones, and so we have a Gantt table of the project to order them.

2 State of the Art

The idea of the app, as it is, I have not seen it in any projects online. But, in the separate parts of it, There have been some developments online and technical papers.

Starting with the part of AI, There are papers related to food dish classification. This means, passing a picture of a plate to a neural network to classify the type of dish in the picture, for example, Bite AI [1]. If we go more creative with that, there are apps that do classify the dish and they tell the nutrient information of the dish. We have for example Calorie Mama [2]. If we look at certain implementations using Yolo Neural Network, we do find a neural network trained with Darknet Yolo by Benny Cheung to do food detection. http://bennycheung.github.io/yolo-for-real-time-food-detection.

In order to develop the 3D graphics and handle all the computer-human interaction and AR Capabilities, we will use the Unity 3D Game Engine. We could use Unreal Engine, but Unity is more capable for extended reality applications, having focused on that subject for the past years. As well, we can see internet examples of Yolo [3] working inside of Unity 3D editor and Yolo V2 with AR [4], Yolo is one of the available computer vision neural networks we may use.

The idea of detecting ingredients in an image, we can see this concept by Alexei Evdokimov on Dribble[5], which shows a future possible AR UI that kind of resembles the idea we want to implement, but in a simpler and not as immersive as using a Mixed Reality Headset.

We can see this example of a first-person XR Cooking experience developed by Lauren Cason [6] on Linked-in. I don't know the headset for which the demo was developed, or if the demo is just a mockup and does not use any headset that has real-time capabilities of actually displaying that content over the ingredients.

In [7] they are able to do food segmentation on the different dish ingredients, a pretty interesting thing, but still not what we are looking for, ingredient detection on its natural state. [8] is more interesting, it does detect objects in a densely packed space, like a supermarket shelf. A supermarket shelf can also remind us of a fridge in some way.

3 Methodology

In order to achieve this project, I will work using a waterfall scheme, developing each module one after the other (or concurrently) as individual small projects. I have separated the project in modules for that matter, so that they are kind of independent from each other and then we can join each one. I use the waterfall scheme as it is a one person job project.

In order to track the tasks on each module and establish myself deadlines on each one, I will use the tool Notion [9], that enables the creation of databases information and show that information in different views. These views can range from a Kanban Trello style view to a Gantt table view. [10]

4 Development Process

As discussed in the objectives, this project has three, we can say 'small' projects inside, but we will see that they are not that small. So first we need to start by investigating, training and bench marking the Neural network and get it to work inside unity. Then, develop the XR UI and the several methods of user interaction input. Finally, create our custom API with recipes.

4.1 Researching Neural Networks to do ingredient detection in real-time

The task was to research neural networks for real-time object detection using a camera input. Yolo was chosen due to its efficiency and simplicity. Its predecessors, like R-CNN and its variations, used a pipeline to perform this task in multiple steps. This is not efficient and can't be optimized easily. That is why in Yolo, the neural network only has one step, and it has to only look once, hence its name Yolo (You only look once).

We are not going to extend too much in the full explanation of how Yolo works, it basically, takes an image as input, and it converts into cells. Once it is converted into cells, the input is converted into a simple regression problem, with its loss function and conventional network. You can get more info on how a Yolo works in this article [11].

The current latest Yolo version is 7 [12] which has a great update over its predecessors, but as it is very new, people have not started using it in their developments. That is why, for example, we find the Yolo V5 [13] that is more developed and refined. Given that fact, one of the main tasks would be training both of the Yolo and see which one is better for our needs.

As we saw in the State of the art, we do have examples of Yolo running in Unity 3D, which can give us a certainty that we are on the right track.

But first, before training, we need a dataset, right?

4.1.1 Finding datasets of ingredients

We need to find a dataset that has pictures of ingredients, and more important that those are annotated. Yolo needs pictures

along XML files with annotations of where to find the object in the image. After an exhausting search, We find Ingredient101 [14] and Foodi-ML from Glovo company [15] but they are not what we are looking for. Luckily, changing our search keyword to 'groceries' is where we find a Groceries dataset from Freiburg University at Germany [16]. This dataset has around 5000 images divided in 25 classes or elements. Those are the following ones:

TABLE 1: LIST OF CLASSES THE DATASET HAS. THIS MEANS THE 'INGREDIENTS' WE WILL BE ABLE TO DETECT.

Beans	Cake	Candy	Cereal	Chips
Milk	Nuts	Oil	Pasta	Rice
Chocolate	Coffee	Corn	Fish	Flour
Honey	Jam	Juice	Soda	Spices
Sugar	Tea	Tomato Sauce	Vinegar	Water

Of course, these are not all things available in a supermarket or fridge either all the basic ingredients for cooking the simplest recipe. But, as a starting point to actually make the project go from an idea to a proof of concept, this is a good start. In the future it would be interesting even at the end of the development, to train a newer neural network with a bigger dataset, or the same one with more images.

Aleksandar Aleksandrov at GitHub did part of the job for us and re-did the same dataset introducing annotations [17]. But they are in Pascal-Voc format and Yolo uses a different format for annotations. For that, I did discover the startup Roboflow [18], which gives a web app software that allows you to upload your dataset, watch it, edit it, for example, add more pictures and annotations, and especially, export in the format you want, in our case YoloV7 and YoloV5 format.

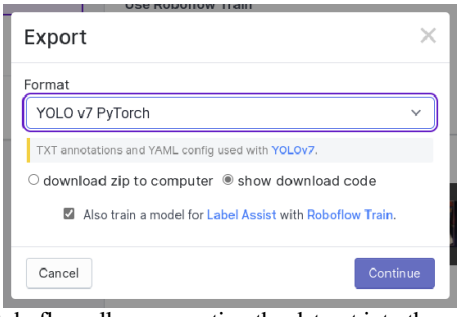


Fig. 1: Roboflow allows exporting the dataset into the annotation format you want.

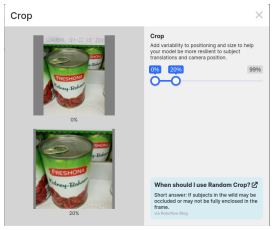
See figure 1 for reference. As well, Roboflow allows Data Augmentation on the go. Data augmentation is the process of making a dataset bigger by adding old but modified pictures, for example rotating them, putting them in black and white, and much more augmentations. More information about it on the appendix.

In our case, we can achieve converting a 5000-picture dataset to a 10.000-picture dataset with it, doing the following augmentations:

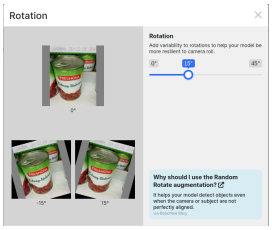
TABLE 2: LIST OF MODIFICATIONS DONE FOR DATA AUGMENTATION ON THE DATASET.

Crop: 0% Minimum Zoom, 20% Maximum Zoom
Rotation: Between -15° and +15°
Shear: ±15° Horizontal, ±15° Vertical

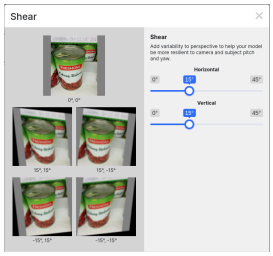
These augmentations can be seen in the following example figure 2



(a) Crop augmentation



(b) Random rotation augmentation



(c) Shear movement done on images.

Fig. 2: Example of the different augmentations done for obtaining more data.

4.1.2 Training the neural networks

While searching for the datasets we can start downloading the YoloV5 [13] and YoloV7 [19] repositories from Github and start looking at how to train them. I did train both neural networks with the Augmented Dataset 2 and 300 epochs (default number) and the input images are of size 224×224.

Yolov5 was trained with the Small architecture that has 7.2 M params and 16.5 Flops. (Details in GitHub repository) and the YoloV7 with YoloV7 middle architecture. Future wise we may try YoloV7 tiny which has less parameters, so it is faster, with a cost on the obtained accuracy.

4.1.3 Comparing the neural networks

Once trained at first we can have a look at model metrics to compare both neural networks. Of course, these are not the direct metrics we need to look into in order to choose which one to use, we will also need to look into how they will perform on our smartphone that will be running them. Next Section will cover that.

The results we get from training Yolo V5 and V7 can be seen in the following figures 3 and 4

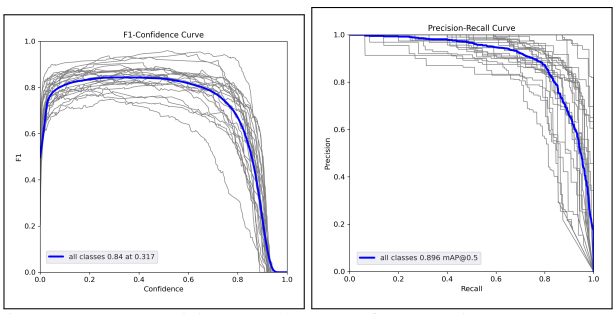


Fig. 3: F1 and precision-recall curve of Yolo V5. F1: 0.84% PR: 0.896%

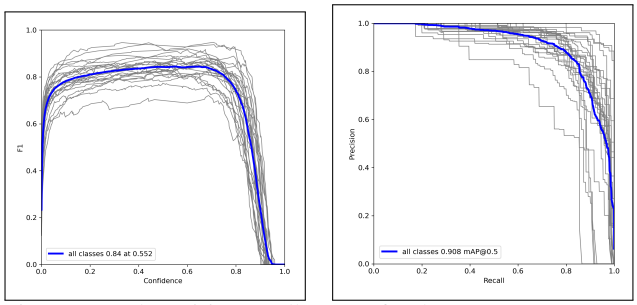


Fig. 4: F1 and precision-recall curve of Yolo V7. F1: 0.84% PR: 0.908%

Having figure 5 as a reference, we have more results that can help us decide. I believe the important ones are the last 4 ones, we see that the graphic has a tendency to go up as we increase epochs, so maybe future wise we could try training for more epochs, like for example 500. On section Results we have proven that.

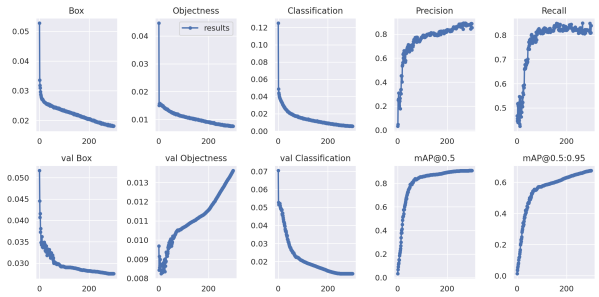


Fig. 5: Results of YoloV7

4.1.4 Introducing into Unity

Inside the Game Engine, we have different ways to make a neural network work inside of it. As we will see I have passed from all of them until we finally got it working on Unity 3D.

4.1.4.1 Exporting to ONNX First of all, I did research on how to introduce neural networks inside Unity 3D software. Unity has developed a library called Barracuda [20] that allows inferencing on the CPU or GPU in real time. In order to make our trained Yolo networks work with the library, they need to be in the ONNX [2].

The use of the Yolo's 'export.py'[3] script presented some issues due to difficulty in understanding the exporting parameters. Initially, the ONNX export was attempted with the options '`--include-nms`' and '`--end2end`', which were thought to be correct based on a review of the code. However, upon viewing the neural network on a website called Netron, it was realized that the resulting ONNX file was not structured as a dictionary where specific values could be accessed. Additionally, attempting to import the ONNX model into Unity resulted in errors due to Barracuda's lack of support.

In light of these difficulties, consideration was given to finding alternative solutions for achieving the working of the neural network in Unity, as a quick and temporary solution was needed in order to progress with the next modules of the project.

4.1.4.2 Temporal solution 1 - Streaming to an external server with WebRTC or Websockets. Given my past experience with web sockets, I developed code quickly that grabbed images from the camera and sent them to a Python code-based server, which would perform inferences on the images using the Yolo-trained network. The server could be either a computer running the Python code or an external cloud. I initially considered using web sockets, which are meant for real-time text but not necessarily real-time images. However, I considered using WebRTC as it is a protocol developed for real-time streaming of images (Used in Zoom, Google Meets, etc...) Despite being interested in learning the WebRTC protocol, I found it would have a steep learning curve and instead chose to continue using web sockets, which proved to be a faster solution. You can see the web socket communication temporal solution in the following video [21] for a better understanding.

Despite this, I was not 100% happy with it, as the project now involves having a second external device, and all that involves. So I decided to close the temporary solution I was working on, which was not completed to 100%.

4.1.4.3 Final solution to make Yolo work in Unity3D To revisit the state-of-the-art, YoloV5 was successfully integrated into Unity by Egor Ulianov [13]. He demonstrated the use of YoloV5 in the 3D engine for QR code detection in images. It is important to note that the output structure of any trained Yolo network remains consistent.

Analysis of Egor's solution revealed two mistakes in our approach. Firstly, incorrect parameters were used in the export of the Yolo model to ONNX, leading to a suboptimal result. The option '`--include-nms`' and '`--end2end`' was used, which is intended for use with the ONNX-Runtime python library [4]. Upon removal of these options, a significant improvement was observed, with the .onnx file being imported into Unity without any compatibility issues.

Secondly, we attempted to extract data from the model using examples from YoloV2 and its structure. YoloV2 returns inference results as a matrix of values. The following figure 6 highlights the differences in output structure between Yolo V2, V5, and V7.

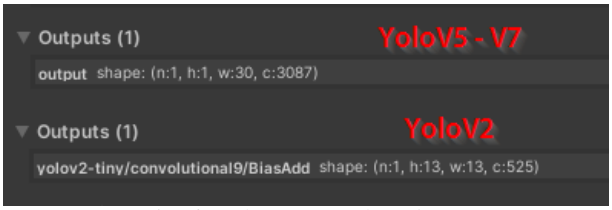


Fig. 6: Yolo V2 vs Yolo V5 V7 .

First, some context to understand the figure 6. Unity Barracuda works with a structure called Tensor [5] (see Figure 7 for reference)

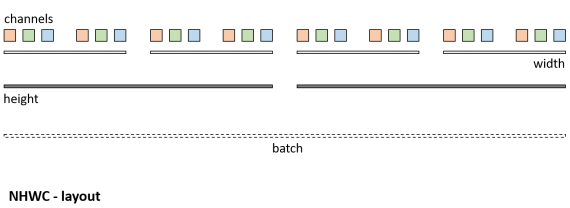


Fig. 7: Tensor structure.

The ONNX model data is represented as a tensor in Unity. In our implementation, only one image is processed at a time, resulting in a single batch output.

[2]The Open Neural Network Exchange (ONNX) is an open-source artificial intelligence ecosystem of technology companies and research organizations that establish open standards for representing machine learning algorithms

[3]https://github.com/WongKinYiu/yolov7/blob/main/export.py

[4]https://onnxruntime.ai/

[5]https://github.com/Unity-Technologies/barracuda-release/blob/release/1.0.0/Documentation /TensorHandling.md

The output structure of YoloV2 is a matrix of shape 13x13, while YoloV5 and Y7 return a list of 3087 or "c" channels. The first 5 elements of each channel in the list represent the X, Y, Width, Height, and box-confidence of the bounding box. The remaining 25 elements are the class probabilities, given that the model was trained on 25 classes[1]. An incorrect processing approach was initially taken, but this was corrected by considering the correct indexing of the output list.

4.1.4.4 Final solution? Not that fast. Barracuda library works, but due to the vast variety of layers and modules an ONNX module can have, it is not able to technically support all ONNX Models, even though the engine does not output any kind of error.

This has been a source of frustration for me over the past several months. Not only frustration for me but also for Barracuda users on the internet, we can see that in the Issues posted on the Github Repo [6].

This lack of support in our neural network made us get a lot of false positives and completely wrong results compared to doing inference with other libraries outside of unity, for example ONNX-Runtime.

Even, at times, it made me doubt if the problem was really in Unity 3D or the training and our neural network. An extensive analysis helped me to realise that we were talking about the first option, although as we will see later, the trained model does not have 100 accuracy.

To make all work, the fastest-to-deploy solution we can do is return to the temporal solution 1 4.1.4.2 discussed before. This time, we can improve it by implementing Sockets and threads, which allow us to send more data reliably, and then having a secondary web socket that sends the inference result in real-time. See figure 8 for a diagram reference of the solution.

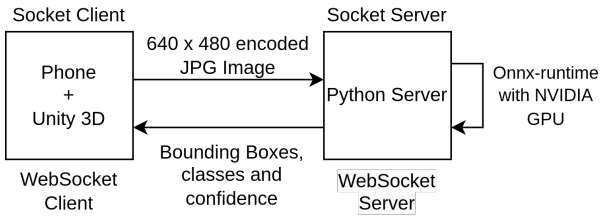


Fig. 8: Diagram of the solution.

Despite being a solution, it has computational cost, as encoding an image in jpeg is good for when you are saving images spodarically but not in real-time. That is why we are using Threads, so we can do the encoding and then write to the socket in a separate thread that is not the main one. Of course, future wise, other solutions for sending images to an external server should be considered, such as WebRTC, discussed in 4.1.4.2.

4.2 Testing YoloV7 vs YoloV5 on Smartphones

The end device that will run the neural network is the Smartphone. It is that way, as the Glassear headset works by putting your smartphone inside of it. Future version of the headset will have built-in electronic and cameras, but in its current state it will not be available before finishing this project, thus we are developing the app to run on Smartphones.

With the magic of programming, the headset has a work in progress SDK (more information on future sections) so porting the project to the next model of headset would be painless and possibly only having the need to change some options.

[6]https://github.com/Unity-Technologies/barracuda-release/issues

To test the different models, first we need to have a look at the pipeline running on the app and doing the inference, as shown in Figure 9

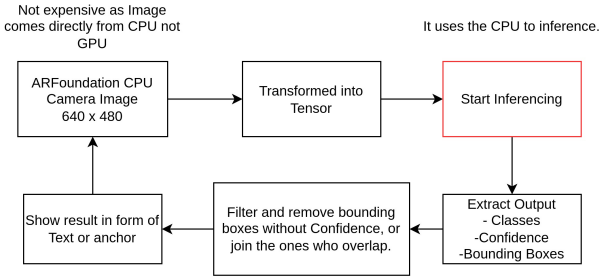


Fig. 9: Pipeline of the inference running in Unity App.

At the end, this part of the pipeline is the one where latency depends on the model. (Red one) The other parts of the pipeline, processing the extracted data, creating the bounding boxes, or creating anchors (explained in section 5.3.1) is the same no matter which neural network we use. As well, camera images are extracted directly from CPU.

Disclaimer: CPU Usage

The app uses a positioning library called ARFoundation, explain in more detail in the following sections. This library, in the android side, is not compatible yet with Vulkan, instead only works with OpenGL.
Barracuda library only does inference on the GPU if it runs on Vulkan architecture, so this means the project is using the CPU for inferencing, wich is slower and did not allow real-time inference.
The newer solution, as inference is done on a server, preferably one with a GPU, the inference is much faster, in addition to the fact that the client does not inherit the computational costs of inferencing.

4.2.0.1 Testing Yolo's with CPU Inference For the simply task of choosing between Yolo V5 and V7, we have calculated the time it takes to infer using the CPU with Unity Barracuda.

```
using (var tensor = TransformInput(picture, IMAGE_SIZE, IMAGE_SIZE))
{
    var inputs = new Dictionary<string, Tensor>();
    inputs.Add(INPUT_NAME, tensor);

    // create a variable with start time
    var watch = System.Diagnostics.Stopwatch.StartNew();

    yield return StartCoroutine(worker.StartManualSchedule(inputs));
    var output = worker.PeekOutput(OUTPUT_NAME);

    // create a variable with end time
    watch.Stop();

    var elapsedMs = watch.ElapsedMilliseconds;
    Debug.Log("Elapsed time: " + elapsedMs + "ms");

    debugText.text = "Elapsed time: " + elapsedMs + "ms";
```

Fig. 10: Snippet of code that does the inference using the Barracuda library.

Figure 10 shows the code that inferences and process the data of the Yolo NN, where we can put a timer to measure latency.

Speaking of Yolo NN, it can be made as a child from a parent that has the base functions. This way, if in the future we decide to switch to another Yolo network or any other type of NN that is not a Yolo, We simply need to create a script that follows that base functions, and the other modules of the pipeline will interact with it.

We have to have in mind that the Barracuda library function StartManualSchedule is Async, so it is more difficult to measure

the time as if it was synced and we know where it starts and ends exactly. As well, it does not allow technology to run in real-time, as we do have some delay on each inference, but it does not affect our primary goals.

Figure 3 shows the latency measurements calculated running the app for more than 5 seconds for both Yolo V5 and V7 respectively:

TABLE 3: TABLE OF DIFFERENT LATENCIES EXTRACTED FROM INFERENCING WITH YOLOV5 AND YOLOV7 IN SIX DIFFERENT EXECUTIONS

Model	Latency #1	Latency #2	Latency #3
Yolo V5	9,8576 ms	9,9876 ms	9,8051 ms
Yolo V7	12,4007 ms	12,4981 ms	12,5102 ms

Model	Latency #4	Latency #5	Latency #6
Yolo V5	9,9901 ms	9,9105 ms	9,9210 ms
Yolo V7	12,5508 ms	12,4183 ms	12,5132 ms

We can see in figure 4 the average time taken to infer with each model:

TABLE 4: AVERAGE LATENCY OF BOTH YOLO VERSIONS V5 AND V7

Model	Yolo V5	Yolo V7
Average	9,9901 ms	12,5508 ms

We get an increase in latency with the Yolo V7 but looking at the performance metrics such as precision and recall, Yolo V7 is slightly better than V5. So for our purposes, we will focus on improving the neural network future-wise using the Yolo V7 architecture.

At the first iteration of the code, the latency was much higher, arriving at a point where the app did block itself and crashed. It was because one part of the pipeline, specifically the one processing and creating the bounding boxes, had poorly implemented and unoptimized code that created a bottleneck.

5 IMPROVING THE NEURAL NETWORK

To improve the NN we would start by first, looking at another possible dataset, even though as we saw, we searched on the whole internet and didn't find much.

Despite that, we can start by doing more data augmentation, as by the current testing of the Yolo Network, a good prediction is very dependent on the background context and the class itself.

For example, the dataset class coffee does not mean pictures of coffee beans, instead, it means coffee boxes sold in supermarkets, as that is how the NN was trained. In fact, all pictures of the dataset were taken in a supermarket so the background is different than for example our house. So we need to focus on making the dataset context agnostic.

As reading article[7] part of the solution is to synthetically augment many many random backgrounds in the training data. So that the network learns that there is no relationship between the object in question and its context.

[7] https://qr.ae/prRvoA

5.1 Generating backgrounds

We can start by looking at kitchen background images online. We find this indoor scene dataset by MIT [8] that contains indoor images of kitchens and other buildings.

What we are able to do can be resumed in these steps:

1. Get the kitchen background image
2. Read the original 224×224 image and extract the Objects
3. Paste those objects on the background on a random position and scale.
4. Generate images these images for every class.

What we achieve is a way to generate however pictures we want, with different backgrounds related to our source problem, the kitchen. At the same time, those background images are of a bigger size (640×640), so the pasted objects look smaller. This might help our neural network detect better the objects that are far away. As we can generate pictures, we can balance the dataset by generating or removing more images.

Figure 11 shows an example of a generated picture following the steps above:

Fig. 11: Background is a kitchen, and we have the objects on top of it.

5.1.1 Transfer Learning

First, we are able to obtain more than 20.000 images where the background images are based on kitchens from the MIT dataset. However, as a way to learn and improve our model, We can also add backgrounds from the kitchen where we do live, this way we can make the model improve by letting him grasp a bit of knowledge of our problem domain.

First I did train for about 16 epochs with the original 20.000 images mentioned before. Once we get the weights of the Yolo Model, we can generate another dataset of 20.000 images with our own background and do start training again from those weights.

This means the model has learned with internet kitchen backgrounds and our own background images. In addition, we can make a dataset of pictures where we have more than one class on the same picture, for example by using the 'Mosaic' technique, see Figure 12 below for reference.

Finally, we re-train the model again with that new dataset and the weights from our last transfer learning. As you may see, we are transferring learning between model weights, having a final model whose weights have been calculated from all our different learnings. That is why it is called transfer learning, as we are transferring different learning to our models.

5.1.2 Results

We see that we have a subtle improvement in detecting objects that are not that near to the camera, as well as an increase on our performance. For example, metrics obtained at 4 or 3 were obtained when training for 300 epochs, in our case now, Figure

[8] https://web.mit.edu/torralba/www/indoor.html

Fig. 12: Mosaic works by taking four source images and combining them together into one.

13 shows us that being at epoch 138 we have surpassed those numbers.

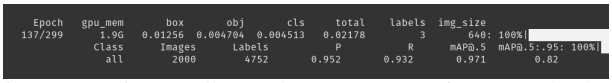


Fig. 13: Metrics are all over the 90 percent at fewer epochs than 300.

Given that we do not have a powerful training hardware infrastructure, we have been training for more than 4 days to arrive at this state, but looking at the way it is going, arriving at 300 epochs will make our model better than before.

Part of that long time to train is that we have increased the number of images, as we saw in 2 we got an output of 10.000 images, and with our technique now, we are having more than 20.000 images.

Now that we have the neural network, we can extract data such as bounding boxes of where objects are located, which brings us to the next step, developing the mixed reality interface for the headset and designing the human-computer interaction. Too see a demo of just the neural network working, you can watch this simple AR example develop [22] (It is also available in the Appendix)

5.2 XR Input and Interface, programming and designing the 3D Graphics.

The neural network was one of the biggest parts of the project, in coalition with Graphics. We need to design and program the 3D interface the user will see when using the headset, and the interaction with that interface.

I started by developing the display of the detected object names over the objects themselves, as seen in [5] and the IKEA 3D Scene from the Fight Club movie [23]. This was achieved through the use of SLAM (Simultaneous Localization and Mapping) [24], an algorithm that creates a real-time map of the environment and calculates the pose of the camera in the scene. SLAM is also used in autonomous vehicles and drones. In terms of AR/MR, we have ARCore [25] on Android, ARKit [26] on Apple, and ARFoundation [27] on Unity, which is a wrapper over both ARCore and ARKit, as well as others such as Microsoft Hololens headset [28].

5.2.1 Spawning detected class names in user's reality. Anchors

When you open the app, you need to start looking around as the SLAM algorithm starts looking for feature points that can allow him to track reality and start creating meshes. Once that, the Yolo network would have started also scanning. So, whenever a new class is detected, we first look if that class is already spawned, if not we do spawn that class name. How do we spawn that? In ARFoundation we have something called Anchors and raycasting.

Whenever we receive a Bounding box of where a class is located, that bounding box lives in the 2D Space of our screen. With raycasting, we 'throw' a ray that comes out from the bounding box coordinates, and so whenever the raycast hits a place of reality where we can attach objects, we do 'Anchor' as we would do with a ship, our 3D Object with the text being the name of the class detected. With current phones, if they are not depth enabled, the distance from the phone to the real object is 'estimated', with future versions of the headset as we increase in cameras, we can calculate that distance exactly.

See Figure 14 below on a graphical representation on how it works:

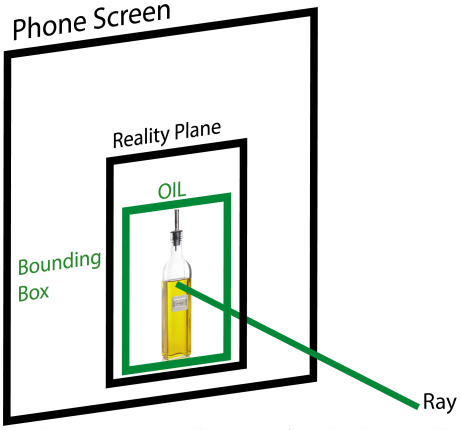


Fig. 14: We have our Bounding Box for the detected ingredient, we can raycast to it and so we get the position on the calculated reality 3D plane where we anchor the Name OIL on top of it.

5.2.2 Introducing the Glassear SDK and designing the UI

In the end, this project is built upon bricks. We have the neural network brick that needs to be joined with another brick to create a house, or as we might call it, GlassFood, our project. To help in the development of the app for the Glassear headset, an SDK (Software development kit) or as I like to call them, toolboxes, has been in development over the past years. It is a toolbox in the sense taht it contains all the essential tools that can help us join the different bricks, the neural network with the hand or voice interaction and the 3D Interface. As well, it contains the different settings needed for the app to work on the Glassear headset.

5.2.2.1 SDK Inputs and Camera image Access The SDK enables different services and inputs, those being:

1. Hand Tracking
2. Plane Tracking (AR Foundation SLAM)
3. Voice interaction
4. Gaze Interaction (Head Tracking Gyroscope Sensor)
5. Spatial Sound

The first two services require access to camera frames in order to perform the necessary computations. The frames are accessed through the SDK's API, which is optimized to not impact the GPU's performance, which is important for rendering graphics. To receive the camera frames in Unity, the scripts must inherit from the ImageWorker class from the SDK.

The hand tracking module is one example that receives and processes camera frames in real-time to detect the landmarks of both hands, including their position and rotation, which is useful for developing interactions. The SDK had to be updated and tweaks made to make integration easier, resulting in an improvement in performance from 30 fps to 60 fps due to the implementation of asynchronous hand tracking inference.

5.2.2.2 UI Design As a first idea of a 3D interface, I did recall an old project back in the day for inspiration and that can be seen in this video [29] (It is also available in the Appendix) The background would be your reality but as it is recorded inside the Unity editor you see the default Unity background. I wanted to make something different and better, which is why in the process of developing the UI, I made different iterations, as each one had its benefits, but None seemed to be all right for our purpose.

We want a UI that follows these rules:

1. User Friendly, easy to interact with using the available inputs. (Gaze, Hand, Voice)
2. Minimal, even though we have our whole reality as a room for our UI, I don't want the user to feel like he is in a Minority Report Movie[9], and that he might get overwhelmed if a lot of information is shown at the same time.
3. Adequate to reality, once the user puts the headset on, the UI should be the more blended with reality as possible. Something that in the pasts years has been called 'Skeuomorphism' [10], if you look at the UI Desing of the iPhone 4, icons always tried to resemble 3D real-life objects for people to understand them easily.

5.2.2.3 UI Iterations These are the different UI iterations I did before achieving the final one. Figure 15 demonstrates the first iteration, where recipes are in a row of cards:

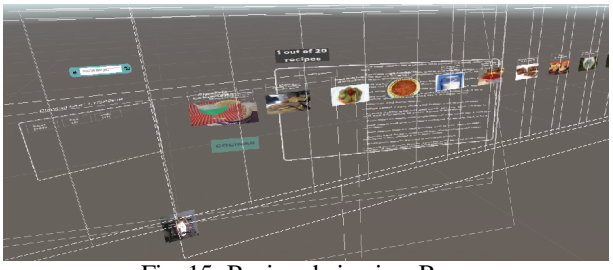


Fig. 15: Recipes being in a Row.

UI Iteration 15 would make the user have to move a lot on the space, moreover it is not adequate to something we would see in our reality. That is why a second iteration was needed, see Figure 16 for reference:

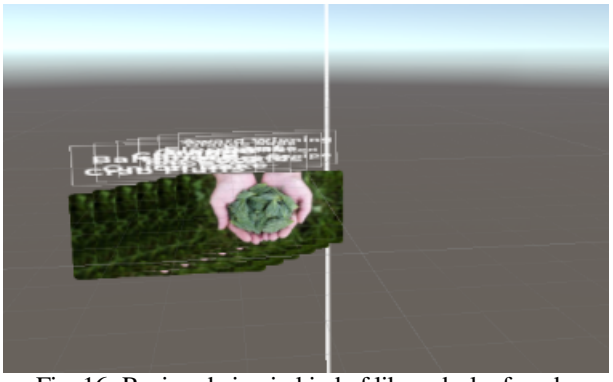


Fig. 16: Recipes being in kind of like a deck of cards.

Figure 16 shows a UI iteration that does adequate to reality, as people are used to playing card games. Still, it was not very user-friendly, as the cards do collapse with each other, making it hard for a user to understand how to interact with them and read their content, and might also feel overwhelmed. As well, that kind of interaction with the cards would have been difficult to make 100% efficient.

[9]https://www.youtube.com/watch?v=jG2iaU-JVhI

[10]https://www.engadget.com/2010-05-07-minority-report-ui-designer-john-underkoffler-talks-about-the-fu.html

After these two iterations, I thought of actually making some UI that resembled an actual recipe website, or almost any website these days (See Figure 17 for reference). The content is shown in a grid of images and when the image is observed, more information would be shown about that recipe:

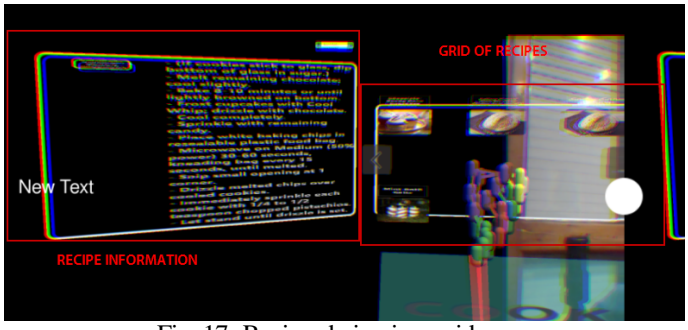


Fig. 17: Recipes being in a grid.

This one did adequate to reality and I like it until I did find out it might not be that user-friendly. First, a user might get in trouble as the objects on the grid are near. Second, we need to make lots of head movement, as we need to first look at the recipe in question, and then look to our right to see the whole information about the recipe.

So, in the end, I went into the roots of cooking, even before the Internet existed. People used recipe Books. What a more adequate UI to the reality than one which resembles a book page? As shown in Figure 18, we can make the UI be like a book page:

Fig. 18: Single recipe and all its information in a single 'Paper-page' like graphic.

The ingredients that the user has, are shown in green color, as they are in theory available for use.

As you may have seen in every design, there is a common UI element, a search bar. Figure 19 below demonstrates the idea of such an element, being it a way to give visual feedback to the user on what is happening at every time:

Fig. 19: The search box has different 'chips' that recall us of Google's Material design guideline [11]

For example, it shows the detected ingredients from the Neural Network, and because it resembles a search box, it hints to the user that those values on that box will be the query for searching the recipes. The box is filled automatically as the neural network

starts giving results, and the search box follows the user around as it moves its head.

We can also do animations with it, for example as we see in Figure 20, when the recipes API query is being done it serves as a loading visual feedback, like loading icons on web browsers.

Fig. 20: The search box bar fills itself with different colors as something is loaded or for example, a new output is added to the query.

In this image above 20, red color is used for example if the API query outputs any error, those being for example, no internet available, so it is not able to contact the external server, among others.

When a new ingredient is added to the query, or everything loaded successfully, it can be turned to green, so the visual feedback to the user is that something 'Good' happened, see Figure 21 below for reference:

Fig. 21: Text shows for a couple of seconds so the user can read it.

Figure 22 also hints us how we can also show text feedback on it for a couple of seconds, so user is able to read it correctly:

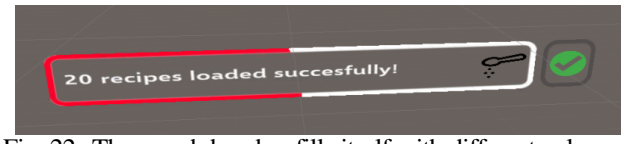


Fig. 22: The search box bar fills itself with different colors as something is loaded or for example, a new output is added to the query.

5.2.2.4 UI Interaction As I mentioned before we do have certain types of inputs. These vary from Hand Input to Gaze input. These are developed to enable interaction inside the mixed reality experience, for example, using your hands to interact with the elements or your gaze.

When using your gaze, for example, it means that it detects where are you looking. Human people when we would like to select something, we look at it by moving our heads and eyes.

This enables our first type of interaction for the project, looking at the UI to interact with it, as we can see in the following Figure 23

(a) Button highlighted when being look at it.

(b) Button in idle state when not looking at it.

Fig. 23: We have a cursor that follows our head movement. With its visual feedback, the user knows they are looking at something that is interactable.

Regarding hand interaction, at the moment, we are not able to 'physically' touch the UI with our hands (due to time constraints and that it is a very difficult task). But as our UI is a book page, we can, for example, put the book page position to one of our palm centers, and so move it around with our hand.

As ARFoundation is creating 'planes' for each feature in reality, for example, a wall, we could move the book page and 'attach' it to the wall, for example in case we are cooking and the book page is in our sight, we can move it to somewhere it does not disturb.

5.3 Recipes API

The project requires the integration of recipe information to provide users with the desired results. For this purpose, several online APIs and services were researched, including Edamam [30], Spoonocular [31], The Meal DB [32], and Recipe Puppy [33]. However, these services have limitations on the number of free requests and require payment for additional usage.

To overcome these limitations, a large recipe dataset was found from MIT, UPC, and Qatar Universities, which comprises over 1 million recipes [34]. Although the information provided by this dataset is limited, it offers the advantage of having a closed and controlled database that can serve as a ground truth for the project.

To make the most of the recipe dataset, a server was developed using Python and the FastAPI [35] library. The server has access to a PostgreSQL database where the recipes were inserted, allowing for fast and efficient search queries. FastAPI was chosen for its simplicity, efficiency, and compatibility with OpenAPI specifications for APIs. With Python STMP Libraries, we are able to be notified with an email once the 1 Million recipes are uploaded.

PostgreSQL was chosen as the database for storing the recipes due to a large number of entries. The server and database were containerized using Docker, making it easy to start and stop. To run the server, a VPS (Virtual Private Server) is used.

5.3.1 Natural Language Processing

The dataset contains the ingredients needed for the recipes simply stated as full sentences, Ex: '2 cups of water and olive oil'. For the correct development of the project in time, we can leave them as it is.

Nevertheless, future wise we can train another model that does Natural Language processing so we parse the ingredients sentences, and we obtain each value independently. For example, having in reference Figure 24 below, our sentence '2 cups of water and olive oil' could be output as a JSON object with each piece of information separately:

```
{
    "quantity": 2,
    "unit": "cups",
    "ingredients": ['water', 'olive oil']
}
```

Fig. 24: Ingredient sentence parsed using NLP to extract quantitative data.

We obtain quantitative data that allow us to show the information in a better way from the UI Perspective, and also improve the search of recipes, as we have the exact token of the ingredient used in the recipe. We have this example of a python library that does Natural language processing [36].

6 Conclusion

Despite being a project with big milestones, we achieved to implement a challenging Computer Vision problem into a next-generation device and UI Interface with innovative human-computer interaction. However, we have a big room for improvement, first of all with Deep Learning, as we have seen in section Improving the Neural Network.

The UI can be also improved, first by making it look sharper and faster, and improve the Human-Computer Interaction by adding more examples of natural input such as your hand. Right now, the hand is only used for moving objects around, but it could be used for example for interacting with the buttons. API-Wise, we could make it faster, so loading times are smaller, and also use some kind of Natural language processing to process the Ingredients information, and extract the correct information separately. Also, have more information about the recipes, such as more nutritional information or also videos, so we can show them on the app.

Regardless of that, we have matured a project that can grow into something bigger than its current state, thanks to its modularity. We can improve current modules as we have stated before, and we can add more modules to add more functionalities to it, giving it more value to our end-user, the extended reality world, and future partners, such as food manufacturers or restaurants.

Acknowledgements

First and foremost, I would like to thank my advisor Felipe Lumbreras, for their guidance, encouragement, and invaluable feedback throughout the entire process.

I would also like to acknowledge the support of my family and friends, who were always there for me, encouraging me to keep pushing forward even when the going got tough.

Finally, I would like to thank the organizations and institutions that provided me with the resources and opportunities to undertake this work, such as Roboflow, Unity 3D, and the creators of the Groceries and 1 Million recipes dataset.

References

[1] “Bite ai - food recognition api.” [Online]. Available: https://bite.ai/food-recognition/

[2] “Calorie mama food ai - food image recognition and calorie counter using deep learning.” [Online]. Available: https://www.caloriemama.ai/

[3] “egor-ulianov/yolov5-unity: Unity barracuda yolov5 android app.” [Online]. Available: https://github.com/egor-ulianov/yolov5-unity

[4] “Localize 2d image object detection in 3d scene with yolo in unity barracuda and arfoundation.” [Online]. Available: https://github.com/derenlei/Unity_Detection2AR

[5] “Recipes searching ar app by alexei evdokimov for wimble on dribbble.” [Online]. Available: https://dribbble.com/shots/6479913-Recipes-Searching-AR-App

[6] L. Cason, “Future of cooking? only in #ar!” [Online]. Available: https://www.linkedin.com/feed/update/urn:li:activity:7003239845045116928/

[7] X. Wu, X. Fu, Y. Liu, E.-P. Lim, S. C. H. Hoi, and Q. Sun, “A large-scale benchmark for food image segmentation,” 2021.

[8] E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, *Precise Detection in Densely Packed Scenes*, 2019. [Online]. Available: https://github.com/eg4000/SKU110K_CVPR19

[9] “Notion – the all-in-one workspace for your notes, tasks, wikis, and databases.” [Online]. Available: https://www.notion.so

[10] “Gantt table in notion.” [Online]. Available: https://glassear.notion.site/ab91c4ba5c014f478400cc561b025f21?v=530a689d84f847f7a5bc7e469ee3dc61

[11] “Understanding yolo — hackernoon.” [Online]. Available: https://hackernoon.com/understanding-yolo-f5a74bbc7967

[12] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” 7 2022. [Online]. Available: https://github.com/WongKinYiu/yolov7

[13] “ultralytics/yolov5: Yolov5 in pytorch ¿ onnx ¿ coreml ¿ tflite.” [Online]. Available: https://github.com/ultralytics/yolov5

[14] M. Bolaños, A. Ferrà, and P. Radeva, “Food ingredients recognition through multi-label learning,” 7 2017.

[15] D. A. Olóndriz, P. P. Puigdevall, and A. S. Palau, “Foodi-ml: a large multi-language dataset of food, drinks and groceries images and descriptions,” 10 2021.

[16] P. Jund, N. Abdo, A. Eitel, and W. Burgard, “The freiburg groceries dataset,” vol. abs/1611.05799, 2016. [Online]. Available: https://arxiv.org/abs/1611.05799

[17] “Andreas eitel - autonomous intelligent systems (annotated for yolo).” [Online]. Available: https://github.com/aleksandar-aleksandrov/groceries-object-detection-datase

[18] “Roboflow: Give your software the power to see objects in images and video.” [Online]. Available: https://roboflow.com/

[19] “Wongkinyiu/yolov7: Implementation of paper - yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.” [Online]. Available: https://github.com/WongKinYiu/yolov7

[20] “Introduction to barracuda barracuda 1.0.4.” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html

[21] B. M. Ruiz, “Unity websocket camera streaming to python server (temporal solution for tfg).”

[22] ——, “Example yolo v7 working in unity 3d for ingredient recognition - youtube.” [Online]. Available: https://www.youtube.com/shorts/Hbw-wbt9khs

[23] “Fight club ikea scene - youtube.” [Online]. Available: https://www.youtube.com/watch?v=exL51n3py6g

[24] “Simultaneous localization and mapping - wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[25] “Crea nuevas experiencias de realidad aumentada que combinen perfectamente con el mundo digital y el físico — arcore — google developers.” [Online]. Available: https://developers.google.com/ar

[26] “Augmented reality - apple developer.” [Online]. Available: https://developer.apple.com/augmented-reality/

[27] “Marco de trabajo ar foundation de unity — software de realidad aumentada para desarrollo multiplataforma — unity.” [Online]. Available: https://unity.com/es/unity/features/arfoundation

[28] “Microsoft hololens — tecnología de realidad mixta para empresas.” [Online]. Available: https://www.microsoft.com/es-es/hololens

[29] B. M. Ruiz, “Example mixed reality interface for watching recipes - youtube.” [Online]. Available: https://www.youtube.com/watch?v=0_vqRrM__cU

[30] “Edamam - food database api, nutrition api and recipe api.” [Online]. Available: https://www.edamam.com/

[31] “spoonacular recipe and food api.” [Online]. Available: https://spoonacular.com/food-api

[32] “Free meal api — themealdb.com.” [Online]. Available: https://www.themealdb.com/api.php

[33] “An ingredient based recipe search engine - recipe puppy.” [Online]. Available: http://www.recipepuppy.com/

[34] “Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images.” [Online]. Available: http://pic2recipe.csail.mit.edu/

[35] “Fastapi.” [Online]. Available: https://fastapi.tiangolo.com/

[36] T. Strange, “Ingredient-parser: A tool to parse recipe ingredients into structured data.” [Online]. Available: https://github.com/strangetom/ingredient-parser