Controlled Image Generation for Reflecting Eating Habits in Virtual Avatars

Bachelor Thesis

Department of Computer Science
OST - University of Applied Sciences
Campus St. Gallen

Spring Term 2025

Authors: Aziz Hazeraj, Thashvar Uthayakumar
Advisors: Prof. Dr. Mitra Purandare, Prof. Dr. Christoph Gebhardt
Date: 13.06.2025

Abstract

This thesis explores the use of AI-based image generation to visualize potential physical changes resulting from adherence to personalized meal plans provided by the Smart Eating Platform. The central goal is to generate realistic image transformations that reflect anticipated changes in body composition, thereby enhancing user engagement and motivation.

To identify suitable image generation techniques, we conducted manual testing of several models and selected two open-source methods, ControlNet and Null-text Inversion, for deeper evaluation. Both were systematically assessed by generating several hundred images using different combinations of input images, prompt templates, and parameters. This evaluation led to the identification of an optimal parameter set for each model.

During the course of the project, OpenAI released the GPT-4o image generation model. Although introduced too late for inclusion in the full evaluation pipeline, it was informally tested and showed superior performance in both realism and fidelity to the original image. As a result, GPT-4o was integrated into the final system alongside the two open-source pipelines.

The full solution consists of a modular system architecture with a Python-based FastAPI backend and a React.js frontend. The backend pipeline handles parameter validation, converts meal plans into descriptive textual prompts (termed ”reflection in appearance”), and generates corresponding images using the selected image model. The system supports both OpenAI’s GPT-4o and Qwen3 as language backends, and allows users to select between the three image generation pipelines based on their preferences or technical constraints.

A user study with nine participants was conducted to evaluate image quality and prompt adherence. GPT-4o emerged as the most reliable and well-rated model overall. ControlNet outperformed Null-text Inversion in average ratings, although with greater variability. These findings validate the decision to include multiple generation backends, providing both high-fidelity results and privacy-conscious alternatives for local execution.

In summary, this thesis presents a flexible, user-configurable pipeline for meal plan–driven image transformation, contributing a novel motivational tool to the Smart Eating Platform.

Management Summary

This thesis explores how AI-generated images can be used to visualize the physical effects of personalized nutrition plans, as offered by the Smart Eating Platform. The goal was to develop a visual feedback system that helps users stay motivated by showing them what they might look like after following their selected meal plan.

To achieve this, we developed a prototype that lets users upload an image, select a meal plan, and choose both a language model and an image generation model. The system then generates a realistic image showing the potential physical transformation, based on the selected inputs.

Three different AI image generation models were integrated: ControlNet, Null-text Inversion, and OpenAI's GPT-4o. Each model offers different advantages—such as realism, adherence to instructions, or local execution, allowing users to select the option that best fits their needs or privacy preferences.

The project was carried out within a 16 week timeframe and included research, experimentation, technical implementation, user evaluation and documentation. Despite the limited time, a fully functioning end-to-end system was delivered.

User testing with nine participants showed that GPT-4o provided the most consistent and realistic results. However, the inclusion of open-source models ensures that the system remains flexible and usable even without internet access or external dependencies.

Overall, the solution adds value to the Smart Eating Platform by giving users a more engaging and motivational experience, helping them stay committed to their dietary goals.

Contents

Chapter 1

Introduction

This thesis builds on recent developments in AI-based image generation and personalized health applications. While nutritional platforms offer precise dietary guidance, they often lack intuitive or emotional feedback. By combining structured dietary input with realistic, AI-generated visual projections, this work aims to bridge that gap, providing users with a motivating and personalized view of potential physical change.

1.1 Initial Situation

Recent advancements at the intersection of artificial intelligence (AI), computer vision, and personal health tracking have led to new possibilities in digital wellness applications and photorealistic image synthesis. Generative models, particularly Generative Adversarial Networks (GANs) and diffusion models, have demonstrated exceptional capabilities in producing high-fidelity, realistic images of human faces, including tasks such as age progression, emotional modulation, and facial style transfer [1], [2]. Simultaneously, the growth of personalized nutrition and fitness applications, such as MyFitnessPal, Noom, and Cronometer, underscores a societal shift toward proactive health management. These platforms enable users to track dietary intake, macronutrient distribution, and caloric expenditure, offering a data-centric view of personal wellness.

One of our academic advisors is the owner of a smart eating platform that exemplifies this convergence. The platform provides personalized meal plans tailored to individual users' dietary goals, preferences, and physical activity levels. It dynamically adjusts recipes and nutritional recommendations based on user feedback, creating a highly individualized dietary experience.

Despite the technical sophistication of such platforms, they typically offer feedback in abstract numerical forms, like calories, macronutrients, and target weights. While this data is clinically useful, it often lacks intuitive appeal. Research in behavioral science and health communication suggests that visual and narrative feedback significantly enhances user mo-

tivation, adherence, and emotional engagement in wellness interventions [3], [4]. Yet, few platforms offer personalized visual projections of how adherence to a given meal plan might influence the user's physical appearance over time.

This thesis explores the potential of controlled image generation, conditioned on individualized meal plans and user profile data, as a novel approach to bridging this gap. By transforming a user's portrait image to reflect projected physical outcomes of dietary behavior (e.g., weight change, muscle development, or improved vitality), such a system can offer an intuitive and emotionally resonant complement to traditional health data.

1.2 Problem Statement

As users follow their personalized plans on the Smart Eating Platform, visualizing the potential physical effects of their dietary choices can serve as a powerful source of motivation. By generating realistic previews of anticipated changes, the system encourages continued engagement and helps users stay committed to their health goals.

Although the foundational technologies for image generation and nutritional analysis are independently mature, their integration poses a number of significant challenges. One of the central issues lies in semantic alignment: translating a meal plan, typically formatted as structured text or JSON, into meaningful conditioning variables for image synthesis requires careful modeling. Nutritional properties such as caloric intake, protein density, and activity levels must be mapped to physiologically plausible visual changes, such as alterations in facial adiposity or muscular definition.

Moreover, physical transformation is inherently a temporal process. The body does not change instantaneously in response to diet; instead, these changes occur incrementally over days, weeks, or months. Thus, an effective system should be capable of modeling and visualizing this progression through temporally coherent sequences of images. This is particularly important for maintaining user engagement and setting realistic expectations.

Finally, the success of such a tool hinges on the quality of the user interface. Users should be able to upload meal plans and profile pictures easily and preview and download generated images. Accessibility, responsiveness, and transparency are all essential design considerations.

1.3 Objective

This thesis aims to design and develop a controlled image generation system that transforms a user's profile picture based on a given meal plan. The goal is to offer a personalized, interpretable, and temporally dynamic visual representation of potential physical changes informed by nutritional inputs.

To support this goal, the system was developed as an independent web application. Users interact with a standalone frontend that enables uploading of structured meal plans and a

profile image. Upon submission, this information is sent to a backend pipeline that parses the meal plan, summarizes its nutritional properties, and generates the modified images. While this tool is conceptually designed for potential integration with existing smart eating platforms, such as the one operated by our advisor, it is implemented and evaluated here as a standalone system to support modularity, experimentation, and clearer analysis.

The system accepts meal plans in structured formats (e.g., JSON) and uses a language-based analysis pipeline to derive semantically rich interpretations of the nutritional input. Specifically, the raw meal plan is passed directly to GPT-4o, which performs a qualitative assessment of the dietary contents, including estimated caloric surplus or deficit, macronutrient emphasis (e.g., high-protein, high-carb, or low-fat), and inferred lifestyle activity levels. Based on this analysis, GPT-4o returns a natural language summary describing the expected visual changes in physical appearance, such as "increased muscle definition," "gradual fat loss," or "visible signs of improved skin tone."

This textual description of expected physical outcomes is then converted into model-specific prompts for downstream image generation. Depending on the user's choice, these prompts are formatted and routed to one of the supported back-end pipelines: ControlNet, Null-text Inversion, or GPT-4o image generation. Each model offers different advantages. ControlNet provides more consistent image structure and layout, Null-text Inversion tends to better follow the intended transformation described in the prompt (sometimes at the cost of identity preservation), and GPT-4o enables high-quality, natural language-driven edits.

A web-based interface allows users to interact with the system seamlessly. It supports image and meal plan upload, parameter customization, model selection, and preview and download functionality.

The implementation is encapsulated within a modular Python package, supporting extensibility and integration with external platforms. This modularity allows for iterative updates to the meal plan parser, image transformation modules, or user interface. An accompanying API facilitates potential integration into broader smart health ecosystems.

In synthesizing structured dietary data into visually grounded, personalized outcomes, this research contributes a novel tool that supports digital health literacy, fosters self-awareness, and enhances user motivation through AI-driven visualization.

Chapter 2

Background

For our thesis, we require two different types of AI models, one one hand, we need language models. On the other hand, we need models capable of processing a user's image and modifying it based on their past diet. Specifically, the model should take an image and the past diet as inputs and generate a corresponding modified image as output. To achieve this, we can leverage denoising diffusion models, which are well-suited for high-quality image generation and transformation tasks.

2.1 Denoising diffusion models

Denoising diffusion models are a class of generative models used for image synthesis. The generation process begins with a forward diffusion phase, in which noise is gradually added to a clean image over a fixed number of steps, ultimately transforming it into pure noise. This process is governed by a predefined noise schedule. In the subsequent reverse diffusion phase, the model learns to iteratively predict and remove the noise at each step, progressively reconstructing the original image from the noisy input.

Many modern generative image models, including Stable Diffusion, use a mechanism called *Classifier-Free Guidance (CFG)* to control the trade-off between fidelity to the conditioning input (e.g., a text prompt or an image) and the model's creative freedom.

Originally introduced to improve sampling quality in text-to-image generation, CFG modifies the denoising process during inference by interpolating between an unconditional and a conditional output. The level of guidance is controlled by a scalar value called the CFG scale. Higher values enforce stronger adherence to the conditioning input, while lower values allow for more variation and deviation.

Depending on the application, the CFG scale can be applied to different types of conditioning inputs. For instance, in InstructPix2Pix, both an *image CFG scale* (guiding adherence to the original image) and a *prompt CFG scale* (guiding adherence to the textual instruction) can

be adjusted independently.

Tuning these parameters allows users to balance prompt sensitivity with identity preservation or structural consistency, making CFG a critical factor in the behavior and output quality of many image generation pipelines.

2.2 Image Editing with Diffusion Models

To start off, we looked for base models, which might be able to fulfill our needs. Specifically, we looked at 3 different models: InstructPix2Pix [5], ControlNet [6] and Null-text Inversion [7].

2.2.1 InstructPix2Pix

InstructPix2Pix is a diffusion-based image editing framework designed to modify images based on natural language instructions. Unlike traditional generative models that require explicit editing masks or guidance signals, InstructPix2Pix fine-tunes a diffusion model to interpret and apply changes directly from text commands. By training on paired data of source images, edited images, and textual descriptions, the model learns to perform a wide range of edits, from simple color adjustments to complex object transformations. Its ability to generalize across various image domains makes it a versatile tool for interactive editing applications, reducing the need for manual intervention in image editing.

2.2.2 ControlNet

ControlNet enhances the controllability of diffusion models by incorporating additional conditioning inputs such as edge maps, depth maps, and keypoints. This method allows fine-tuned structural preservation during image generation, making it particularly useful for applications requiring precise control. By introducing trainable control layers that influence the diffusion process without disrupting the pretrained generative capabilities, ControlNet ensures that outputs adhere closely to user-specified constraints. This framework significantly improves the practicality of diffusion models for real-world design and editing tasks, offering a balance between creative flexibility and structural accuracy.

2.2.3 Null-text Inversion

Null-text Inversion is a technique designed to enhance image editing capabilities within diffusion models by refining the text embedding process. Instead of relying solely on direct textual prompts, this method optimizes the null-text embeddings, the parts of the model that define what remains unchanged, allowing for more controlled and localized edits. By aligning the diffusion process with the original image's latent space, Null-text Inversion ensures that modifications affect only the intended areas while preserving overall consistency. This approach is particularly beneficial for tasks requiring fine-grained alterations, such as

facial expression adjustments or background modifications, making it a powerful tool for high-fidelity image refinement.

2.2.4 OpenAI GPT-4o

OpenAI's GPT-4o image generation model introduced a novel approach to image generation based on an autoregressive transformer architecture [8]. Unlike traditional diffusion models, which iteratively refine a noise vector to converge on an image, autoregressive models generate images token by token in a sequential manner, modeling pixel or patch dependencies similarly to how language models generate text.

This architectural difference leads to several practical advantages. Autoregressive models tend to excel at preserving global coherence and structural alignment with the input, including facial identity and background consistency. They are also more capable of producing exaggerated or expressive visual changes in response to prompts, making them particularly effective in tasks like facial editing, character transformation, and scene conditioning. However, these more pronounced transformations are not always desirable, especially in applications where subtle, physiologically realistic changes are preferred.

2.2.5 Newer and Other Diffusion Models

Numerous other models are available online—for example, Flux—but we deliberately narrowed our focus to a curated set of models that we assessed as both high-performing and practical for our use case. Notably, during the midpoint of our project, OpenAI announced a new image generation model that significantly outperformed most existing open-source alternatives.

2.2.6 Related Work

Recent advances in generative AI have led to a range of methods for controllable image editing and appearance manipulation, many of which are relevant to this thesis. While no prior work has addressed our specific goal, generating future appearance predictions based on meal plans, there are several related domains that inform our approach, particularly in facial editing, attribute-based transformation, and conditional image synthesis.

Attribute-based facial editing A number of early works focused on editing human faces by altering specific attributes such as age, weight, or facial expression. One notable example is *AttGAN* [9], which employs an attribute classifier to guide the generation of facial features while preserving identity. Similarly, *StarGAN v2* [10] supports multi-domain image-to-image translation and allows users to modify facial attributes such as age or skin tone with high visual fidelity. These models introduced the idea of localized, semantically meaningful edits, but typically rely on fixed attribute sets and do not support free-text conditioning.

Identity Preservation in Image Editing Identity preservation has emerged as a crucial challenge in personalized image editing tasks. FaceShifter [11], for example, offers a high-fidelity face-swapping approach that uses a two-stage architecture to explicitly decouple identity and pose information. Its design enables realistic and occlusion-aware swaps, achieving strong identity retention even under extreme conditions. Another significant contribution in this area is DreamBooth [12], which extends diffusion-based image generation through subject-specific fine-tuning. Given a small number of input images, DreamBooth adapts the model to recognize and reproduce unique identity features, allowing for personalized text-to-image generation. Although computationally more intensive, it delivers high accuracy in preserving facial characteristics across diverse prompts and contexts.

These methods illustrate different strategies for combining semantic control with identity consistency and underscore the importance of balancing realism, expressiveness, and faithfulness in image-based transformation tasks. Our work complements this landscape by comparing multiple open-source and proprietary models in terms of prompt adherence, identity preservation, and applicability in health-oriented use cases.

2.3 Large Language Models

Large Language Models (LLMs) are sophisticated neural architectures trained on massive amounts of textual data. These models exhibit advanced capabilities in understanding, generating, and transforming human language, making them indispensable tools for tasks such as summarization, translation, code generation, semantic reasoning and much more. Within our image generation pipeline, LLMs play a critical role in interpreting user prompts and converting them into structured, machine-interpretable summaries. In our project, we are using two large language models: GPT-4o from OpenAI and Qwen3 [13] hosted on the LLMHub.

2.3.1 OpenAI

Our primary language understanding component is the GPT-4o model developed by OpenAI. As a state-of-the-art multimodal LLM, GPT-4o demonstrates exceptional performance in natural language understanding and generation across diverse contexts.

2.3.2 LLMHub

To enable efficient inference in computationally constrained or cost-sensitive scenarios, we utilize the Qwen3-30B-A3B-GPTQ-Int4 model hosted on LLMHub. This model is a 4-bit quantized implementation of the original Qwen3-30B-A3B architecture—a Mixture-of-Experts model with 30.5 billion parameters, only 3.3 billion of which are dynamically activated per task. The quantized variant maintains the original model's 131k token context window while significantly reducing computational resource requirements, making it suitable for efficient local deployment and scalable inference in environments where cloud-based solutions are impractical or costly [13].

Chapter 3

Requirements

In this part of the documentation, we will define our requirements for the software part of this project.

3.1 Functional Requirements

The functional requirements define the core capabilities and behaviors the system must exhibit to fulfill its intended purpose. These requirements specify how the system should respond to specific inputs, perform particular tasks, and interact with users and other systems.

Each requirement specifies the following details:

- ID: the ID of the Functional Requirement, unique to each FR
- Requirement: The description of the requirement itself.
- Priority: The priority level of the FR
- Input Type: What is the input / How does the requirement start?
- Output/Result: What is the result / output of the requirement?
- Acceptance Critera: What does it need to fulfill, to be declared as done.

3.1.1 User Interface

The following functional requirements are about the possibilities of our User Interface (UI).

ID	FR 1.1
Requirement	The UI must accept a meal plan.
Priority	High
Input Type	JSON file
Output/Result	Meal plan is shown in the meal plan textfield.
Acceptance Criteria	Meal plan is shown inside of the meal plan textfield and is used when calling the API.

Table 3.1: Detailed View of Functional Requirement FR 1.1

ID	FR 1.2
Requirement	The UI must accept a user's picture (image file).
Priority	High
Input Type	Image file (JPEG/PNG)
Output/Result	The image is shown in the image field.
Acceptance Criteria	The image is saved in-memory and used when calling the API.

Table 3.2: Detailed View of Functional Requirement FR 1.2

ID	FR 1.3
Requirement	The UI must be able to trigger a generation using its parameters and show the result.
Priority	High
Input Type	Click on generate button
Output/Result	The reflection in appearance and the transformed image are shown.
Acceptance Criteria	As soon as the generation is finished, the results are shown to the user.

Table 3.3: Detailed View of Functional Requirement FR 1.3

ID	FR 1.4
Requirement	The UI must allow users to preview and download the modified images.
Priority	High
Input Type	Click on resulting image (Triggers download)
Output/Result	The transformed image is downloaded.
Acceptance Criteria	As soon as the generation is finished, the result will be shown and the image is downloadable via clicking.

Table 3.4: Detailed View of Functional Requirement FR 1.4

ID	FR 1.5
Requirement	The UI must not allow users to use Null-text inversion and LLMHub in combination.
Priority	High
Input Type	-
Output/Result	-
Acceptance Criteria	The UI does not allow the usage of LLMHub with Null-text Inversion in any way.

Table 3.5: Detailed View of Functional Requirement FR 1.5

3.1.2 Prompt generation

The prompt generation requirements define, in what way the meal plan must be processed inside of the Python package in the Prompt generation part.

ID	FR 2.1
Requirement	The package must summarize the meal plan into a few key sentences.
Priority	High
Input Type	Meal plan data
Output/Result	Summary text
Acceptance Criteria	The summary is generated and used in the next step of the prompt generation.

Table 3.6: Detailed View of Functional Requirement FR 2.1

ID	FR 2.2
Requirement	The package must use the summary of the meal plan to create image model specific prompts.
Priority	High
Input Type	Meal plan summary
Output/Result	image model specific prompt
Acceptance Criteria	The s

Table 3.7: Detailed View of Functional Requirement FR 2.2

ID	FR 2.3
Requirement	The package must use the selected LLM.
Priority	High
Input Type	LLM selection
Output/Result	Correct model is chosen.
Acceptance Criteria	The model, which was chosen in the LLM selection, is used for the whole prompt generation.

Table 3.8: Detailed View of Functional Requirement FR 2.3

3.1.3 Image Generation

The requirements regarding image transformation are related to the image generation step in our Python package pipeline.

ID	FR 3.1
Requirement	The system must use AI-based image processing to modify the user's profile picture based on the given prompt
Priority	High
Input Type	Image, Image model specific Prompt generated from the prompt generation step
Output/Result	Transformed image
Acceptance Criteria	A transformed image is returned.

Table 3.9: Detailed View of Functional Requirement FR 3.1

ID	FR 3.2
Requirement	The system must use the selected image model by the user.
Priority	High
Input Type	Image model selection
Output/Result	Correct model is chosen
Acceptance Criteria	The image model selection parameter uses the correct image model

Table 3.10: Detailed View of Functional Requirement FR 3.2

3.1.4 Python Package

This part of the requirements defines the functionality of the Python package.

ID	FR 4.1
Requirement	The whole image generation pipeline should be implemented as a Python package.
Priority	High
Input Type	-
Output/Result	-
Acceptance Criteria	When the pipeline is implemented as a Python package

Table 3.11: Detailed View of Functional Requirement FR 4.1

ID	FR 4.2
Requirement	The package takes in and validates the inputs.
Priority	High
Input Type	image, meal plan, model selections and API keys
Output/Result	cleanly saved parameters
Acceptance Criteria	The inputs are validated and saved in an object

Table 3.12: Detailed View of Functional Requirement FR 4.2

ID	FR 4.3
Requirement	The package returns the edited image and a reflection in appearance.
Priority	High
Input Type	all package inputs
Output/Result	transformed image, reflection in appearance
Acceptance Criteria	After generation in the package the reflection in appearance and the transformed image are returned.

Table 3.13: Detailed View of Functional Requirement FR 4.3

3.1.5 API

The API is the part of the backend, that connects the Python package to the frontend. It features several endpoints for most combinations of image model and LLM.

ID	FR 5.1
Requirement	The API should provide endpoints to integrate into other platforms (e.g., the smart eating platform, in this case, it is our frontend), that want to use the Python package.
Priority	High
Input Type	-
Output/Result	-
Acceptance Criteria	There are endpoints to use our Python package.

Table 3.14: Detailed View of Functional Requirement FR 5.1

ID	FR 5.2
Requirement	The endpoints should be segregated using the Interface segregation principle.
Priority	Medium
Input Type	-
Output/Result	-
Acceptance Criteria	There are endpoints segregated using the Interface Segregation Principle.

Table 3.15: Detailed View of Functional Requirement FR 5.2

ID	FR 5.3
Requirement	The API should be able to handle multiple requests by using a queue and processing each image generation individually and sequentially.
Priority	Medium
Input Type	Multiple requests
Output/Result	Results from each request
Acceptance Criteria	The API is able to accept multiple requests, enqueues them, and works off each request, and provides a result for each request.

Table 3.16: Detailed View of Functional Requirement FR 5.3

3.2 Non-Functional Requirements

In this part of the thesis, we will describe the Non-Functional requirements of our software. While defining our Non-Functional requirements we have tried to be either Specific, Measurable or even both if possible

3.2.1 Landing zones

The table 3.17 displays the landing zones for each described zone. Each zone has three different values:

- Minimum: A minimum value, which should be the lowest goal to reach for.
- Regular: An average value, which is ideal if reached, but not the best outcome.
- Outstanding: A high target, which is the best outcome, but not the standard.

Description	**Minimum** (Within)	**Regular** (Within)	**Outstanding** (Within)
Prompt generation duration	10s	3s	1s
Image model transformation duration	2m	40s	20s
Full pipieline duration	2m	1m	25s

Table 3.17: Landing zones for Non-functional requirements

3.2.2 Performance

In this subsection, we will describe the Non-Functional requirements defined and related to performance, as visible in the table .

ID	Requirement
NFR 1.1	One full image generation using the Python package should process and generate a transformed image within the landing zones 3.17 defined for one full pipeline duration for a standard input image (512x512 pixels).
NFR 1.2	In the process of the generation of an image, the subprocess Prompt generation within the Python package, should adhere to the prompt generation duration defined in the landing zones 3.17 average-length meal plan (30 days of meals).
NFR 1.3	The image transformation in the image generation step of the Python package pipeline, should be within the landing zones3.17 defined for a single image transformation measuring only duration used to generate the image using an image model. All three models (OpenAI, ControlNet, Null-text Inversion), which are integrated into the Python package, should adhere to the defined landing zones.

Table 3.18: Non-Functional Requirements for Performance and Efficiency

3.2.3 Accuracy and Realism

ID	Requirement
NFR 2.1	The AI-generated profile pictures should be realistic and visually plausible, avoiding extreme distortions.
NFR 2.2	The system must prevent exaggerated transformations that could mislead users.
NFR 2.3	The transformation should reflect aspects such as: weight gain, weight loss, muscle growth, overall athleticism and for example tiredness (all according to the meal plan constellation).

Table 3.19: Non-Functional Requirements for Accuracy and Realism

3.2.4 Security and Privacy

This part describes the Non-functional requirements related to security and privacy in our software.

ID	Requirement
NFR 3.1	An uploaded image goes through the whole application. When the image is used to start a generation, the image must be used in-memory and must never be stored persistently.

Table 3.20: Non-Functional Requirements for Security and Privacy

3.2.5 Maintainability and Extensibility

ID	Requirement
NFR 4.1	The system should be modular. This means that it should allow and provide good extensibility.

Table 3.21: Non-Functional Requirements for Maintainability and Extensibility

3.2.6 Usability

This section shows all non-functional requirements, related to usability. In our case it is related to the UI.

ID	Requirement
NFR 5.1	The UI should be simple and intuitive, allowing users to upload images and meal plans effortlessly. The generation should be able to started with a simple click and the results should be shown to the user, without the need for effort.
NFR 5.2	The UI should be responsive and work on both desktop and mobile devices.

Table 3.22: Non-Functional Requirements for Usability

3.2.7 Compatibility

The last part of the non-functional requirements include the compatibility of our system.

ID	Requirement
NFR 6.1	The Python package must be compatible with 3.9+.
NFR 6.2	The UI should support major web browsers (Chrome, Firefox, Edge, Safari).

Table 3.23: Non-Functional Requirements for Compatibility

Chapter 4

Exploration

This chapter documents the exploratory phase of the project, in which various image generation models were tested to gain an understanding of their behavior. Before committing to a systematic evaluation framework, we conducted hands-on experiments with a range of architectures to better understand their capabilities, limitations, and suitability for visualizing diet-induced physical changes. These early trials helped shape our model selection, interface design, and parameter configurations for the remainder of the thesis.

4.1 Initial Testing

A main part of this thesis is the evaluation of the main three models: InstructPix2Pix, ControlNet and Null-text Inversion. As an initial step, we conducted extensive hands-on testing with these models to better understand their behavior. These early experiments helped us identify promising directions and informed the design of our later, more systematic evaluations.

4.1.1 Initial Experimentation with Pix2Pix

In the early stages of this project, we conducted exploratory testing using the Pix2Pix framework to evaluate its potential for controlled image transformation based on semantic input. Specifically, we experimented with the InstructPix2Pix model, which extends the original Pix2Pix formulation by conditioning transformations on natural language instructions [5]. Our primary goal was to test whether the model could generate subtle, realistic changes, such as increased muscularity or weight gain, while maintaining the identity and appearance of the original person.

For initial evaluation, we used the publicly accessible demo[1]. This platform allowed us to quickly iterate on prompts and parameters without the overhead of local deployment.

[1] `https://huggingface.co/spaces/timbrooks/instruct-pix2pix`

Users are provided with only two tunable parameters: the *image CFG scale* and the *prompt CFG scale*. The image CFG controls how closely the output adheres to the original image, while the prompt CFG controls how strongly the model follows the textual instruction. We experimented with various combinations, typically varying the image CFG between 1.0 and 2.0, while changing the text CFG between 5.0 and 8.0.

Despite this parameter tuning, we found the model difficult to control in a consistent or meaningful way. Prompts with similar wording often resulted in drastically different outcomes, and the transformed images frequently deviated from the original identity. In some cases, the model produced plausible results, where the output reflected the intended transformation reasonably well, even if facial resemblance to the original subject was only moderately preserved (see Figure 4.1). However, in other cases, such as the prompt ”make him chubbier”, the model generated unrealistic or exaggerated features, such as bloated facial structures or distorted proportions (see Figure 4.2). These issues persisted even with conservative CFG settings.

Figure 4.1: Prompt “make him more muscular” yields an acceptable transformation. Original photo by Charles Etoroma from Unsplash

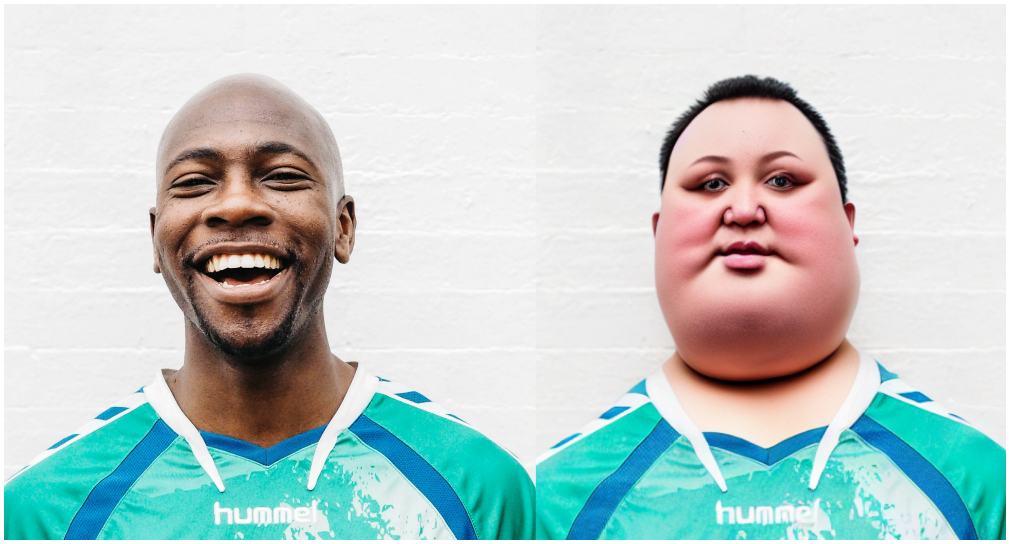


Figure 4.2: Prompt “make him chubbier” results in a distorted and unrealistic image. Original photo by Charles Etoroma from Unsplash

Overall, InstructPix2Pix lacked the precision and identity retention necessary for our use case. The unpredictable prompt response and exaggerated outputs made it unsuitable for visualizing realistic, health-related body transformations. These findings informed our decision to move toward more controllable and consistent architectures, namely ControlNet and Null-text Inversion.

4.1.2 Initial Experimentation with ControlNet

Following our evaluation of InstructPix2Pix, we turned our attention to ControlNet, a more advanced method for conditioning image generation on structured input such as edge maps, depth maps, and poses. To assess its capabilities, we began with the publicly available web demo[2], which offers a simplified interface for prompt-based image generation using a variety of conditioning modes.

Initial results were promising. Although the generated images showed substantial changes from the original inputs, the subject’s core body and facial features were often retained, resulting in outputs that maintained a recognizable resemblance to the source person. This marked a clear improvement over InstructPix2Pix, which frequently distorted identity, facial structure and the overall picture composition.

The demo interface provided a limited number of parameters to control generation. We experimented with different conditioning methods, including Canny, MLSD, Scribble, OpenPose, and Depth. Among these, the *Depth* mode, particularly with the DPT preprocessor enabled, delivered the most consistent and visually plausible results. It preserved important

[2] `https://huggingface.co/spaces/hysts/ControlNet-v1-1`

facial contours and structural details, which are critical for maintaining subject identity during transformation.

However, the web interface constrained deeper experimentation. Many advanced parameters, such as the number of diffusion steps, guidance scales, and prompt weighting, were available only in limited form, and the demo lacked batch processing or reproducibility tools. To overcome these limitations and allow for more fine-grained control, we transitioned to a local deployment using the *Automatic1111 Stable Diffusion WebUI*.

4.2 ControlNet

To gain more granular control over the generation process and overcome the limitations of the online demo, we installed the *Stable Diffusion WebUI* by Automatic1111 and integrated the *ControlNet extension*. This local setup enabled us to fine-tune key parameters, test various ControlNet layers, and experiment with custom prompts and models in a reproducible environment.

Through iterative testing, we adjusted core generation parameters such as the classifier-free guidance (CFG) scale, the number of inference steps, and the adherence to prompt weighting. Among these, the most impactful change was reducing the *Denoising strength* parameter from its default value of 0.75 to approximately 0.35. This adjustment significantly improved the identity preservation of the generated images. At this setting, the outputs retained most of the visual characteristics of the original input image, including background and clothing, while reflecting only the changes specified in the prompt, such as weight gain or weight loss. This was a critical breakthrough in our research, as earlier attempts had failed to achieve both realism and prompt fidelity simultaneously.

In parallel, we explored the impact of different ControlNet conditioning modes—specifically, scribble-based conditioning and depth maps. To better understand how these influenced the visual quality and identity preservation of outputs, we generated paired examples under both settings using identical prompts and generation parameters. The differences were subtle but informative: depth-based conditioning yielded slightly more coherent and realistic facial features (see Figure 4.3, while the scribble-based method introduced minor artifacts, particularly in facial regions (see Figure 4.4. These variations were most noticeable in the expression and symmetry of facial components, even though the overall transformation strength and structure remained nearly identical. The relatively small difference is likely due to the low denoising strength, which limits the influence of the conditioning input. Based on these observations, we opted to use depth conditioning as the default for subsequent evaluations, as it offered a slight advantage in identity fidelity.

Figure 4.3: ControlNet result using depth conditioning. Left to right: input image, depth map, generated output. Original photo by Paguiloumathi from Pixabay

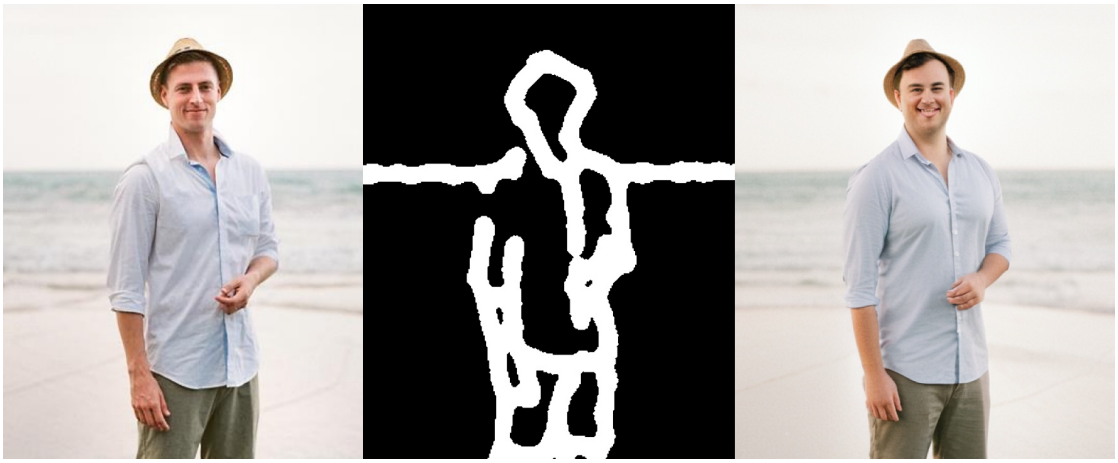


Figure 4.4: ControlNet result using scribble conditioning. Left to right: input image, scribble layer, generated output. Original photo by Paguiloumathi from Pixabay

Despite these improvements, we still encountered a major limitation. While body features and context were now handled effectively, facial realism remained a consistent weakness. Faces often appeared distorted, asymmetrical, or uncanny, detracting from the overall quality of the output. At this stage, we were still using the default *Stable Diffusion 1.5* base model.

To address this, we replaced the base model with *Juggernaut XL v9*, a high-fidelity, realism-optimized model available via Hugging Face [14]. This change had an immediate and no-

ticeable impact. Facial features became significantly more realistic and symmetrical, and the identity of the subject was preserved to a much higher degree. Compared to previous results, the images now displayed both the semantic accuracy described in the prompt and the visual coherence needed for realistic, identity-consistent transformations.

This phase marked the most successful point in our testing pipeline, representing the culmination of iterative improvements to both model selection and parameter tuning. The combination of ControlNet with low denoising strength and Juggernaut XL as the base model delivered the best balance of prompt adherence, identity preservation, and photorealism that we had achieved thus far.

4.3 Null-text Inversion

After completing a series of experiments with ControlNet, we turned our attention to *Null-text Inversion*, a method introduced by Mokady et al. [7] that allows editing real images by leveraging diffusion guidance while preserving key visual features. We aimed to test whether this approach could offer more semantically coherent and identity-preserving transformations than earlier methods.

For initial testing, we used the official Jupyter notebook provided by the authors in their GitHub repository[3], which supports Null-text Inversion as an extension of their Prompt-to-Prompt framework. The workflow requires an input image and a caption describing the scene or subject. Modifications are made by editing specific words or phrases in the caption, which then guide the transformation process. For instance, changing ”a slim man in a blue shirt” to ”a muscular man in a blue shirt” triggers the generator to adjust the body type while retaining other elements of the image.

The results from this notebook were visually plausible and semantically aligned with the prompts. However, identity preservation remained inconsistent. Although some edits were subtle and effective, others introduced noticeable shifts in facial features or proportions. Compared to ControlNet, Null-text Inversion produced more variable outputs with higher degrees of randomness and less structural consistency.

Despite these issues, the method showed enough potential that we planned a systematic evaluation. To do this, we needed the ability to automatically generate a sequence of outputs using different parameter settings. However, automating the Jupyter notebook proved to be a major technical obstacle. The notebook relied heavily on global variables and session-specific states, making it resistant to modularization or scripting. Attempts to rework the notebook into a Python script or structured module were unsuccessful despite significant effort.

As a workaround, we decided to use a simplified Null-text Inversion pipeline developed by one of our advisors. This alternative implementation enabled us to perform batch generation, which was essential for our evaluation process. However, it came with certain limitations.

[3] `https://github.com/google/prompt-to-prompt`

Unlike the original notebook, which supported output resolutions up to 1024×1024 pixels, the advisor's pipeline was restricted to 512×512 resolution. Furthermore, the only parameter exposed to the user was the *classifier-free guidance* (CFG) scale, reducing flexibility during experimentation.

Despite these trade-offs, the pipeline allowed us to proceed with systematic image generation and evaluation. Although we lost some image fidelity and parameter control, the ability to scale the process was essential for carrying out our planned comparisons and performance assessment.

4.4 Prompt Engineering

A critical component of our system is the ability to convert structured dietary data into prompts suitable for image generation. Since the quality and specificity of the prompt have a direct impact on the realism and semantic accuracy of the resulting image, we dedicated a significant portion of our development effort to designing a robust and automated prompt generation strategy.

To this end, we leveraged a large language model (LLM) as an intermediate layer between the user's dietary input and the image generation model. Based on the nutritional characteristics and inferred lifestyle patterns extracted from the meal plan, the LLM generates a textual description — a "reflection in appearance" — that summarizes how the person's physical features might change. This reflection is then reformulated into model-specific prompts, tailored to the requirements of each image generation pipeline.

4.4.1 Keyword Discovery and Iterative Refinement

Before designing the final prompt templates, we conducted extensive hands-on testing to identify which keywords produced the most reliable and semantically appropriate transformations across different models. Early experiments revealed that some adjectives yielded more consistent and realistic edits than others, even when the intended transformation was similar.

For example, prompts containing the word *chubby* tended to produce more plausible weight gain effects than alternatives such as *heavy*. Similarly, combinations like *lean and athletic* outperformed simpler descriptors like *thin* in producing toned, healthy-looking results. These findings emerged from iterative trials using the image generation models with varying prompt formulations.

As a result, we compiled a set of effective keywords that consistently triggered the desired semantic and visual outcomes. These terms were embedded directly into the prompt engineering instructions provided to the language model. By explicitly referencing successful examples, such as "make the person look lean and athletic with vibrant skin" or "make the person look extremely obese and less vibrant", we ensured that the LLM-generated prompts would include these high-impact keywords.

This iterative refinement process improved the stability and quality of the generated images by equipping the language model with both the semantic context and specific keywords needed to produce reliable and visually coherent transformations across different image generation pipelines.

4.4.2 Prompt Structure for ControlNet

ControlNet expects short, descriptive commands that directly communicate the intended visual change. These prompts typically follow a directive format, e.g., “make the person look slightly chubbier” or “make the person look lean and athletic with vibrant skin.” To ensure consistency, we developed a structured instruction that the LLM uses to generate such directives from the intermediate reflection (see Listing 3). Several examples are embedded in the instruction to demonstrate the desired output format and vocabulary.

Example instruction to LLM for ControlNet:

> “You get information on how a person has changed after following a certain lifestyle called ’Reflection in appearance’. Create a concise description of how the person has changed in one precise sentence...”

This approach results in prompt outputs that are concise, semantically aligned with the meal plan, and reliably compatible with ControlNet’s inference pipeline.

4.4.3 Prompt Structure for Null-text Inversion

Null-text Inversion operates differently: it requires a textual caption of the base image and performs localized editing based on minimal prompt modifications. Therefore, instead of generating an entirely new description, the LLM modifies the existing image caption by injecting transformation-related adjectives in front of the subject noun.

For instance, the original caption “a man sitting in a kitchen” might be transformed into “a slightly chubby man sitting in a kitchen.” To guide the LLM in performing these edits with consistency, we created an instruction with examples that demonstrate how to rephrase the caption while keeping sentence length and structure close to the original (see Listing 4).

Example instruction to LLM for Null-text Inversion:

> “Using the base prompt given about a person, replace or extend the sentence using the changes reflecting their appearance. Place the new appearance as an adjective in front of the subject...”

This formulation preserves the spatial and stylistic integrity of the original image while applying the semantic transformation with high precision.

4.4.4 Summary

Both prompt generation strategies were implemented as part of a backend module that automatically selects the appropriate template based on the user's choice of image model. This architecture ensures a seamless and consistent transformation pipeline while minimizing manual intervention.

By combining language generation with visual conditioning, this prompt engineering approach enables the system to produce personalized, meaningful, and visually coherent outputs from structured nutritional data.

4.5 Systematic Evaluation

To move beyond anecdotal testing and subjective impressions, we conducted a structured evaluation of the image generation models under controlled conditions. This phase aimed to assess how different parameter settings influence image quality, realism, and alignment with the intended appearance changes described in the prompts. By generating and comparing a large number of images under consistent conditions, we were able to identify model behaviors, performance trends, and optimal configurations.

4.5.1 Elo Scoring System for Image Evaluation

To rank the visual quality and prompt alignment of generated images, we implemented an evaluation interface based on the Elo scoring system. Originally developed by Arpad Elo for ranking chess players [15], the Elo algorithm has been widely adopted in machine learning to model relative preferences from pairwise comparisons. A notable example is its use in reinforcement learning from human feedback, where it enables effective ranking of agent behaviors without requiring exhaustive comparisons [16]. These characteristics make Elo particularly suitable for our application, where a large number of images need to be compared efficiently and fairly.

Scoring Logic Each image begins with an initial score of 1000. When a user selects a winning image in a battle, the scores of both the winning and losing image are updated based on the Elo formula:

$$E = \frac{1}{1 + 10^{(R_{\text{loser}} - R_{\text{winner}})/400}}$$

$$R'_{\text{winner}} = R_{\text{winner}} + K(1 - E)$$
$$R'_{\text{loser}} = R_{\text{loser}} + K(0 - E)$$

Here, R is the current Elo score, R' the updated score, E the expected outcome, and K a constant that controls sensitivity (set to 32 in our system). The greater the difference in

Elo scores between two images, the more the result is expected — leading to a smaller score adjustment. Conversely, unexpected outcomes result in larger changes, rewarding surprising wins and penalizing unlikely losses more heavily.

Implementation The scoring system is implemented directly in the browser using a React-based frontend. Each image comparison is presented side-by-side, and the user selects the better one. When a selection is made, the interface uses the above formula to update the scores of both images in real time. A persistent leaderboard is maintained locally via browser storage, and the final results can be exported as a structured JSON file, containing scores and match counts for each image.

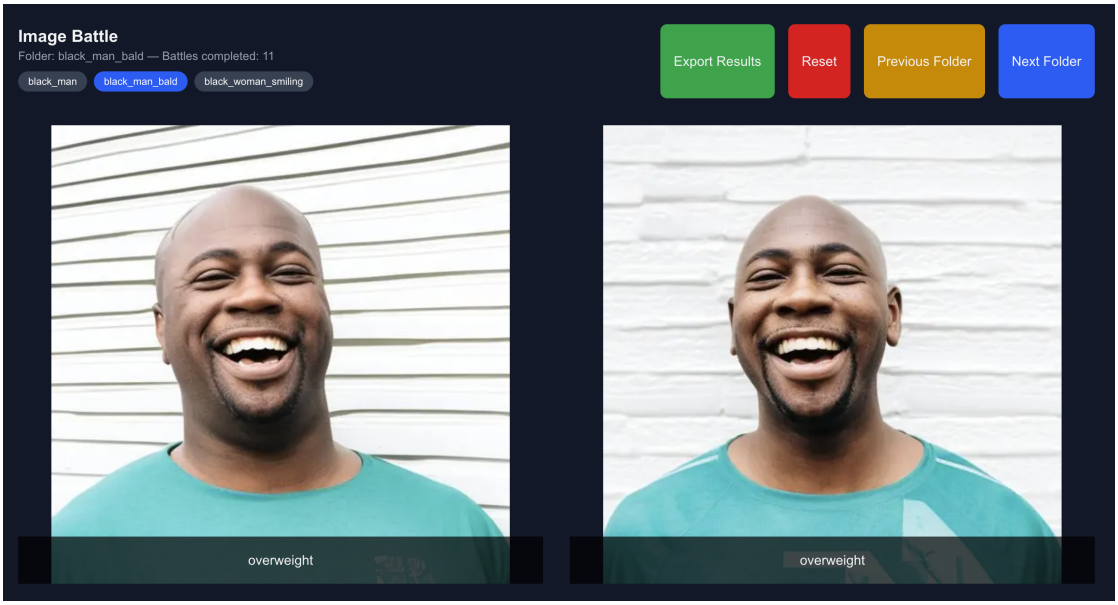


Figure 4.5: User interface for image evaluation using Elo scoring. Original photo by Charles Etoroma from Unsplash

Why Elo? The Elo system is particularly well suited for our use case, as it does not require all images to be compared against each other directly. Instead, a robust ranking emerges from a limited number of comparisons, making it efficient and scalable for large sets of images. It also captures relative differences between images more effectively than absolute scoring systems, which may suffer from bias or scale inconsistency.

This method enabled us to conduct rigorous evaluations while maintaining a lightweight interface. After each rating session, we computed Elo scores per image and then averaged them by parameter set within each rater's results to identify the most effective generation configurations, as seen in Tables 4.1 and 4.2.

4.5.2 Evaluation of ControlNet

To identify the most effective ControlNet parameter configuration for image transformation, we conducted a systematic evaluation of automatically generated images using a structured pairwise comparison interface. The goal was to determine which combination of parameters produced the most visually coherent and semantically accurate results across a variety of prompts and base images.

We defined a total of seven parameter sets, varying key settings such as the classifier-free guidance (CFG) scale and denoising strength. For each configuration, we generated images from six different input portraits using eight transformation prompts. This resulted in a total of 336 unique images (7 weight sets × 8 prompts × 6 images).

The prompts were selected to reflect a range of body transformation scenarios, including:

- athletic lean
- extremely obese
- extremely skinny
- overweight
- rounder belly
- slightly chubbier
- slightly slimmer
- very thin

To evaluate the outputs, we developed a rating interface that allowed us to compare pairs of images side by side. Each of us chose which of the two images better reflected the intended transformation while preserving the subject's identity and realism. This pairwise comparison approach allowed for fast, intuitive evaluations and enabled the emergence of a relative ranking across images.

Over the course of one hour, each of us completed approximately 200 pairwise comparisons, resulting in a total of around 400 comparisons. The outcomes were stored in a structured JSON file that recorded the Elo rating for each image, along with the number of comparisons it had been involved in.

The Elo rating system, implemented directly in the evaluation interface, dynamically updated each image's score based on the outcome of every comparison. Starting from a baseline score of 1000, images gained or lost points depending on whether they won or lost a match, adjusted using a fixed K-factor of 32. This method allowed for relative ranking of images across all prompts and parameter sets without requiring direct comparison of every possible image pair.

After the rating sessions, we aggregated the Elo scores by parameter set, computing average scores across all images belonging to each configuration. The results of both rating

rounds are shown below:

Weight Set	Average Elo Score
1	1031.48
3	1029.88
6	1015.74
5	1008.25
0	1003.44
2	981.00

Table 4.1: Average Elo Scores by Weight Set (Rater 1)

Weight Set	Average Elo Score
1	1012.37
3	1002.25
5	1001.84
6	994.08
2	981.70
0	980.16

Table 4.2: Average Elo Scores by Weight Set (Rater 2)

This structured evaluation provided a solid empirical basis for selecting Weight Set 1 as the default configuration for ControlNet in our system. Notably, both raters independently ranked Weight Set 1 as the top-performing configuration, and the relative ordering of other weight sets showed substantial similarity between the two rating rounds. This agreement across independent evaluations strengthens the reliability of the ranking and suggests that the performance differences observed between configurations are consistent and reproducible.

Figure 4.6 illustrates a representative output using this configuration. The prompt “make him slightly chubbier” was applied to a base image of a person using the Depth ControlNet layer. The result demonstrates a subtle but semantically meaningful transformation, while retaining facial and contextual features of the input image.

Figure 4.6: Transformation using ControlNet with optimal weight set. Prompt: *"make him slightly chubbier"*. Original photo by Charles Etoroma from Unsplash

4.5.3 Evaluation of Null-text Inversion

Following the evaluation of ControlNet, we applied a similar assessment methodology to Null-text Inversion. As described earlier, this approach allows semantic manipulation of real images through textual inversion and guided diffusion [7]. Since the only adjustable parameter in the custom pipeline we used was the classifier-free guidance (CFG) scale, the parameter search space was smaller than in the ControlNet evaluation.

We selected three different CFG values: 1, 2, and 3, corresponding to weight sets 0, 1, and 2 respectively. For each weight set, we generated transformed outputs for three different source images and seven prompts, resulting in a total of 63 images (3 CFG values × 3 source images × 7 prompts).

The same comparison interface used in the ControlNet evaluation was employed again. Each rater independently evaluated images by selecting the one that better reflected the prompt while preserving realism and identity. The outcomes were scored using an Elo-based ranking algorithm, and the final scores were aggregated by weight set.

Weight Set	Average Elo Score
1	1026.11
0	1015.28
2	1004.94

Table 4.3: Average Elo Scores by Weight Set (Rater 1)

Weight Set	Average Elo Score
1	1049.97
0	1016.60
2	993.57

Table 4.4: Average Elo Scores by Weight Set (Rater 2)

In both evaluations, weight set 1 (CFG = 2) consistently outperformed the other configurations, achieving the highest average Elo scores. These results suggest that a moderate CFG value strikes the best balance between guiding the diffusion process and allowing enough variability for effective image transformation.

During this evaluation, we also made a critical observation: framing played a significant role in the quality of the generated outputs. Images that focused more closely on the face produced significantly more realistic and identity-preserving results than those that depicted the full body. Full-body images often suffered from distortion and lacked structural consistency, while close-ups allowed for more accurate and meaningful edits. This finding, initially noted during informal testing, was confirmed during the structured evaluation and informed our later design decisions for prompt and image selection.

While Null-text Inversion still displayed more variability and occasional inconsistencies compared to ControlNet, this evaluation confirmed that, under the right conditions and parameter settings, it remains a powerful and usable tool in our generation pipeline.

4.6 Incorporating GPT-4o

Around the time we completed our systematic evaluation of ControlNet and Null-text Inversion, OpenAI released its GPT-4o image generation model. In our informal tests with the GPT-4o image model, the outputs were not only more realistic and detailed, but also retained significantly closer resemblance to the original input faces compared to both ControlNet and Null-text Inversion. In some cases, however, the visual changes were exaggerated, such as producing extreme muscularity from a mild prompt, which may not always align with applications requiring physiological realism, as illustrated in Figure 4.7.

Reflection in appearance: gains some muscles, more vibrant skin, lean.

Figure 4.7: GPT-4o output showing a detailed but exaggerated transformation. Original photo by Clive Thibela from Unsplash.

Despite recognizing the superior performance of this model, we chose not to include GPT-4o in the formal evaluation pipeline. Since the model does not expose configurable parameters like ControlNet or Null-text Inversion, there was no meaningful way to conduct a comparative parameter study. Additionally, the image generation API became publicly available only toward the end of the project, by which time the evaluation framework and documentation were already near completion.

Nevertheless, we chose to integrate the OpenAI image model into our final frontend implementation as an optional backend model. This provides users with the flexibility to select the generation backend that best suits their needs, particularly in cases where ControlNet or Null-text Inversion do not yield satisfactory results. Including GPT-4o makes the system more flexible and ready for future developments by allowing newer proprietary models to be used alongside open-source and locally run alternatives.

4.7 Temporal Progression

At the outset of this thesis, one of our initial goals was to support temporal image progression, where users could visualize gradual changes in appearance over time based on their meal plan. The idea was to simulate physical transformation at multiple intervals, such as after 30 days, 60 days, and 90 days, by repeatedly applying changes to the same image.

This would allow users to see the potential long-term effects of sustained eating behavior in a more narrative and incremental form.

Our initial approach involved taking the already generated output image from the first transformation (e.g. after 30 days), and using it as the input for the next transformation stage (e.g. to simulate 60 days). This was tested using ControlNet within the Stable Diffusion WebUI.

However, early testing quickly revealed significant limitations with this method. When reusing an already modified image as input, the resemblance to the original person degraded noticeably with each successive transformation. Facial features and other identity-preserving details were increasingly distorted or lost altogether. As a result, the outputs no longer reflected the continuity of identity needed to represent a realistic temporal progression. Instead of simulating the gradual outcome of a consistent meal plan, the changes appeared disconnected and artificial. Because preserving identity was a key requirement of this system, and because this approach introduced compounding artifacts that undermined the goal of progressive visualization, we ultimately decided to discard the temporal chaining approach.

Chapter 5

Architecture

The following sections present the C4 architecture diagrams and the activity diagram for our project, illustrating the system at various levels of abstraction. We have chosen to omit the Level 4 diagram, as it offers limited additional insight beyond what is already conveyed by the other three diagrams. The Context, Container, and Component diagrams sufficiently capture the architectural structure and design rationale of the system.

5.1 System Context

In the following diagram seen in 5.1, we present the system context for our simple UI application. Given the minimal complexity of the application, the context diagram highlights a single primary user who interacts directly with the system. Additionally, two external systems are involved: LLMHub and OpenAI. From LLMHub, the application integrates a language model (LLM), while from OpenAI, it utilizes both a language model and an image generation model.

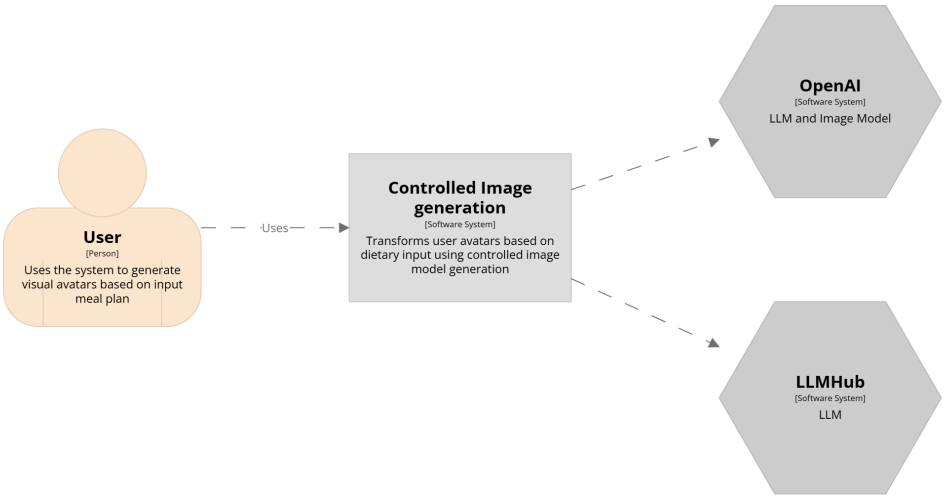


Figure 5.1: System Context

5.2 Container Diagram

The following figure 5.2 illustrates the different containers that comprise our application architecture. The system is primarily divided into two main containers: a frontend built with ReactJS, and a backend implemented using FastAPI. The backend integrates our custom Python package, which encapsulates the core application logic and model interactions.

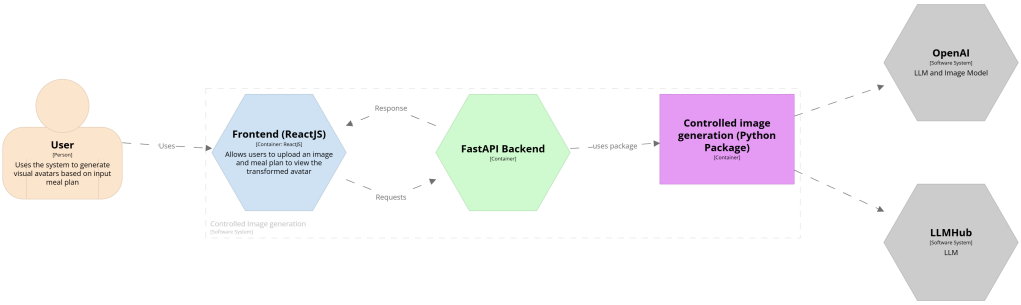


Figure 5.2: Container Diagram

5.3 Component Diagram

The component diagram focuses on the Python package container, which is responsible for generating the edited image. The process follows a pipeline structure: it begins by validating the input parameters, then constructs a descriptive prompt by summarizing the meal plan, and finally uses this prompt to generate the image. This pipeline leverages several AI models, both local and external. All models involved in this process are represented as gray hexagons in Figure 5.3.

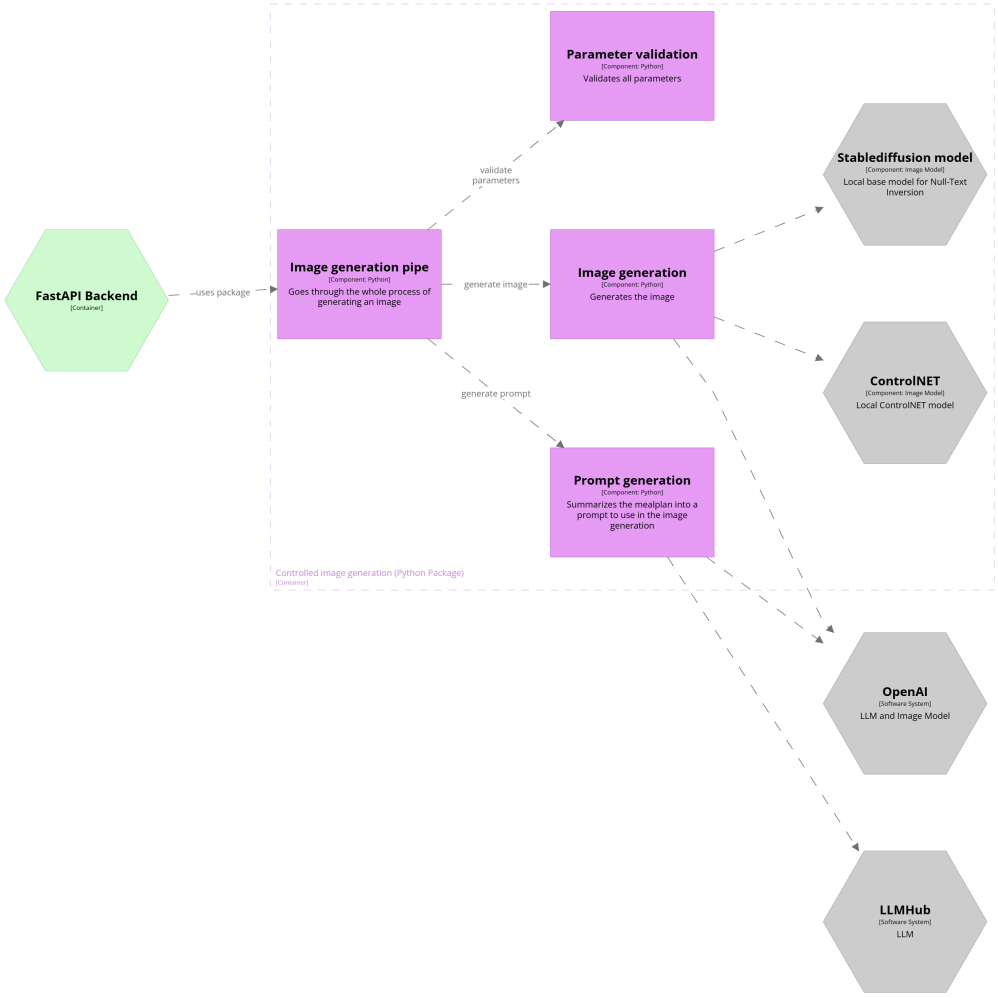


Figure 5.3: Component Diagram

5.4 Activity Diagram

Figure 5.4 illustrates the end-to-end process triggered by a user request, from the frontend interface to image generation and result delivery in the backend package.

The process begins when a user submits a request by providing a meal plan, an initial image, and selecting the desired models. This request is received by the backend API, which enqueues it as a job for background processing. A worker then starts executing the

job.

In the Python package, the first step is validating the input parameters. Afterwards, the prompt generation is started. If the chosen language model is OpenAIs model, the pipeline determines whether a caption of the original image is required. This is only the case, if we use Null-text Inversion as the image model, when we do not require a caption, the LLMHub could also be used. A visual model is invoked to describe the image before the LLM creates a task-specific prompt in case the caption is required. Otherwise, the LLM directly generates a prompt either by compromising the meal plan. If a caption was generated, it will be used to create a prompt similar to the caption. The generated prompt is tailored for the selected image model.

Depending on the chosen image model (OpenAI, ControlNet, or Null-text Inversion), the process branches accordingly. For OpenAI models, the input image is converted into a stream and sent to the API, where null-text optimization may be applied prior to image generation. For ControlNet, appropriate model objects are instantiated and weights are set before the generation step. In the end, all branches lead to the generation of a new image.

Once the image generation completes, the system updates the job result status and makes the result retrievable via polling. The frontend client periodically checks the job status. As soon as the status is on ”completed”, it fetches the result. After fetching the result, it is shown in the user interface and can be downloaded.

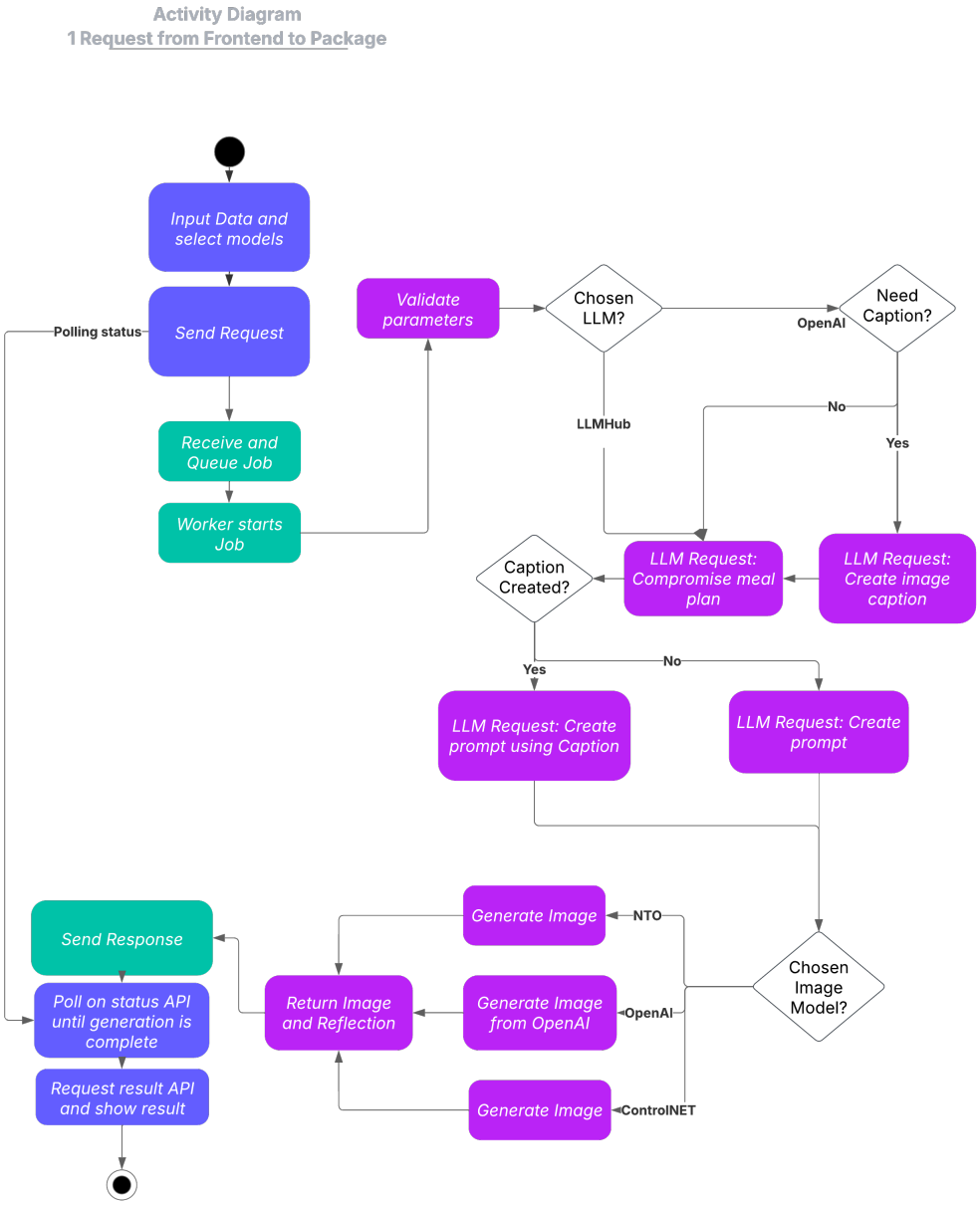


Figure 5.4: Activity Diagram

Chapter 6

Implementation

In this part of the thesis, we will describe our implementation of each part of the whole generation process. There are 3 different main parts, which we have implemented into our system: the ReactJs frontend, the backend using FastAPI connected to our Python package. To see a visualization of our architecture, see 5.2.

6.1 Frontend

The frontend is implemented as a single-page application using ReactJS in combination with the Material UI component library. The primary function of the frontend is to provide an interface for uploading input data, initiating backend API calls, and presenting the resulting output. In the following sections, we will describe the design and the implementation in more detail.

6.1.1 Design

The frontend is implemented as a Single-Page Application (SPA), providing a streamlined and responsive user experience. All input options are positioned prominently on the main interface. Figure 6.1 displays the default view of the user interface (UI) in dark mode, with no inputs provided. As shown in the figure, the interface includes a field for image input, which supports both drag-and-drop functionality and manual file selection. Below that, users can upload a meal plan, provided it is in JSON format. At the bottom of the interface, there are options to select the desired language model (LLM) and image model.

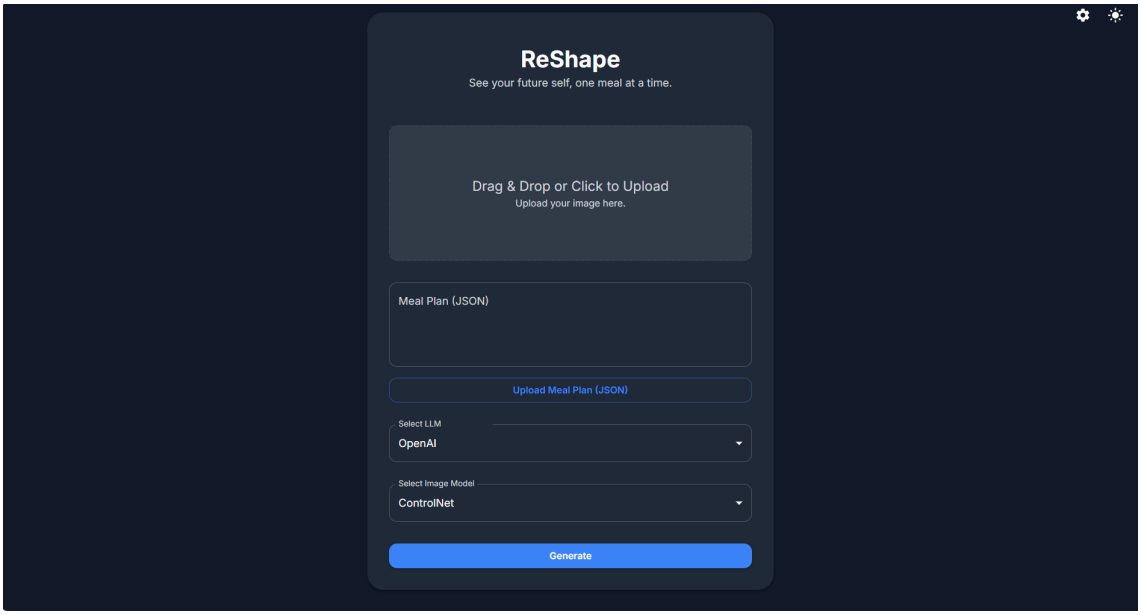


Figure 6.1: Plain UI

The same interface design is also available in light mode, as illustrated in Figure 6.2. The toggle button for switching between dark and light modes is located in the upper right corner and is represented by either a sun or a moon icon, depending on the current display mode.

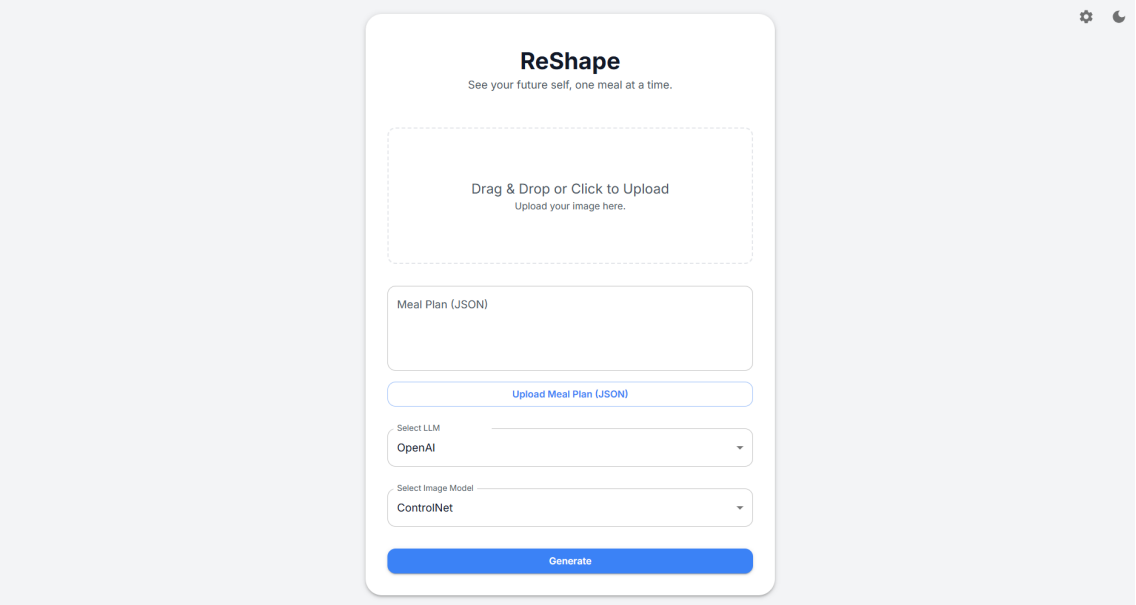


Figure 6.2: Plain UI in light mode

Now we can start to set our inputs as seen in figure 6.4. First of all, we need an image. Then we need to upload our meal plan. Finally, we need to set the preferred models.

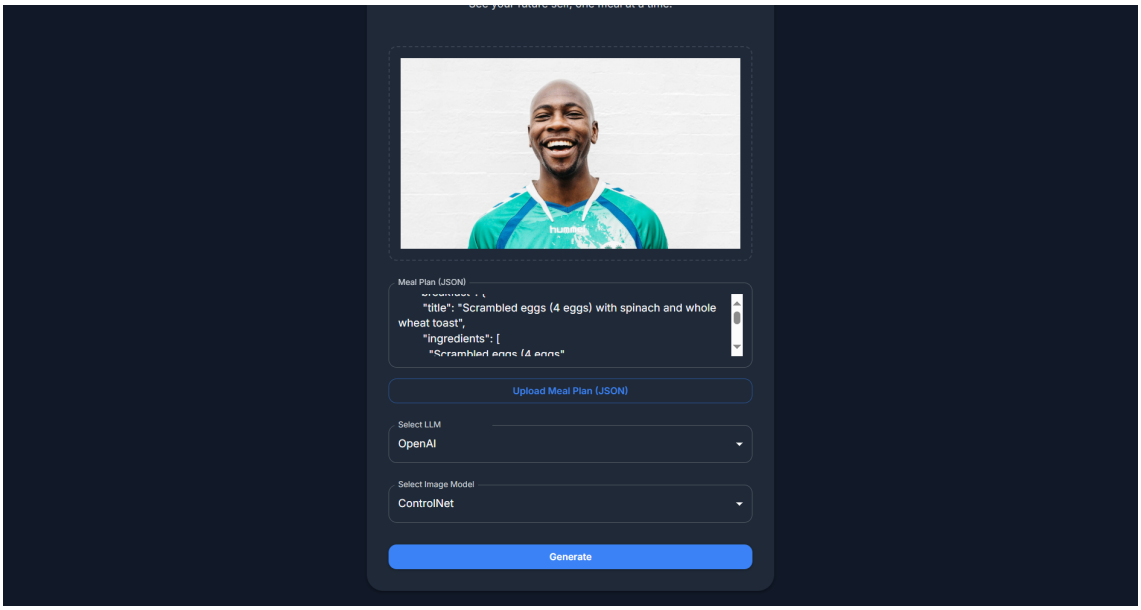


Figure 6.3: UI with inputs; Photo by Charles Etoroma from Unsplash

Before the models can be used, API keys must be configured. This can be done via the settings menu, accessible by clicking the settings icon located in the upper right corner of the interface.

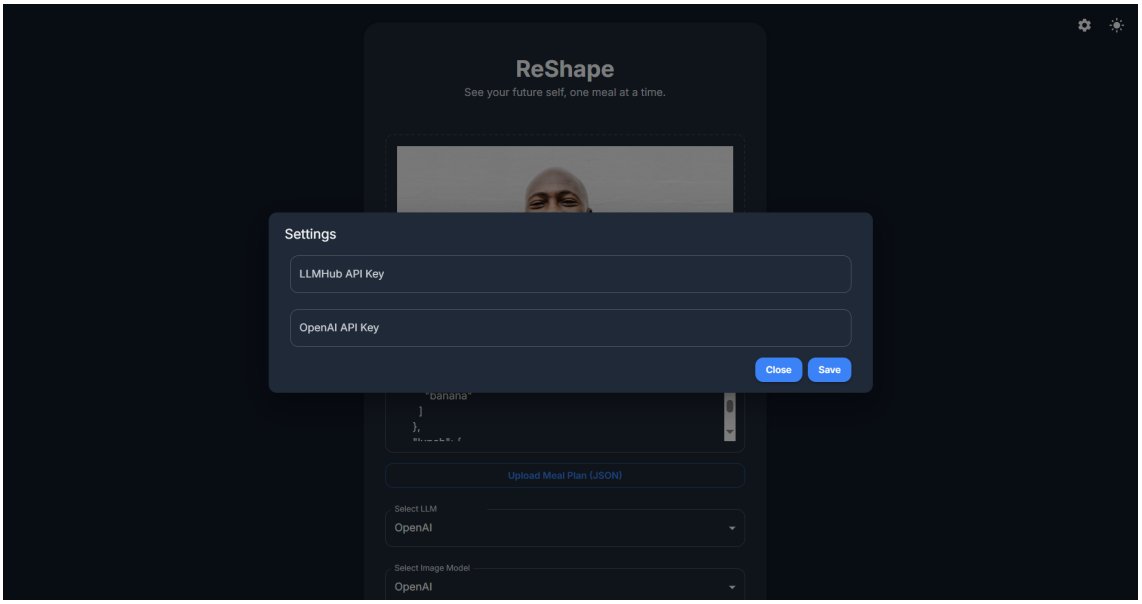


Figure 6.4: UI Settings

Once all inputs have been configured, the generation process can be initiated by clicking the prominent blue ”Generate” button. After the loading phase, the output is displayed at the bottom of the interface, as illustrated in Figure 6.5. The results consist of two main components: a ”Reflection in Appearance” and the transformed image. The ”Reflection in Appearance” provides the user with interpretative feedback on the input meal plan, indicating both the nature of the reflection and the direction of the applied transformation.

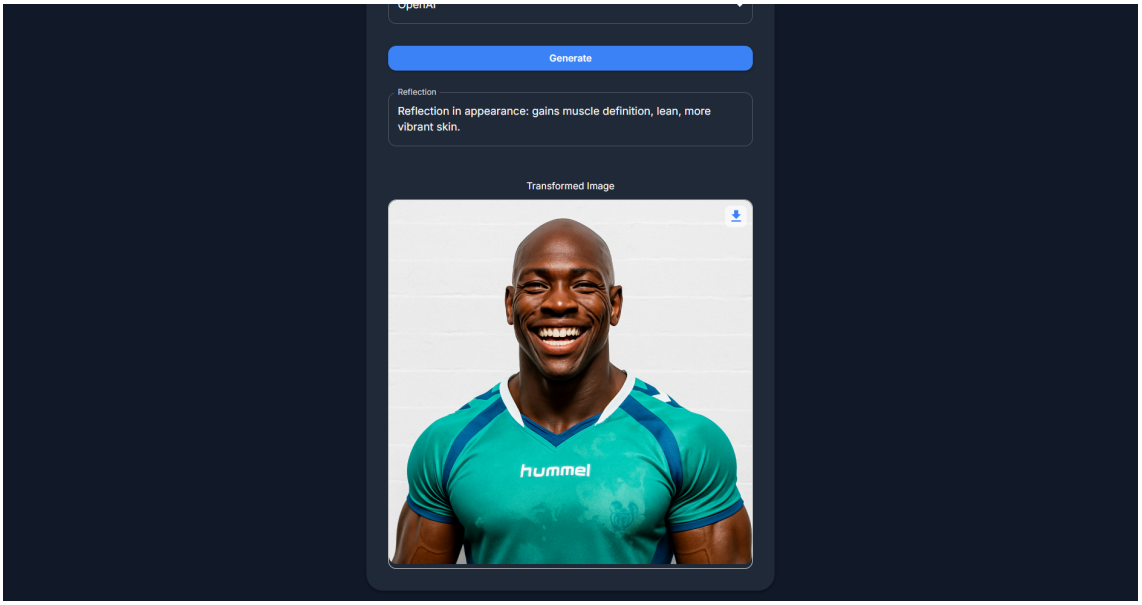


Figure 6.5: UI results; Original photo by Charles Etoroma from Unsplash

6.1.2 Implementation

This section describes the implementation of frontend. Everything is written using ReactJs and complemented with the Material UI component library.

Internally, the application manages several states, including the uploaded image, meal plan content, API keys, selected model, generation progress, and the generated result. State management is handled via React's `useState` and `useRef` hooks.

User inputs are submitted to the API using a `FormData` object containing the uploaded image file, serialized meal plan string, selected model parameters, and authentication keys. Upon submission, the frontend sends a `POST` request to the `/generate` endpoint of the backend API. Once a task ID is received, the application enters a polling loop by repeatedly querying the `/status/{task_id}` endpoint until the task completes or fails.

After successful completion, the frontend issues a `GET` request to the `/result/{task_id}` endpoint to retrieve the base64-encoded output image along with the generated reflection prompt. The response is decoded and stored in the frontend state to be rendered in the view. A download button is provided to allow the user to locally save the resulting image.

The application includes client-side validation for required fields and offers convenience features such as drag-and-drop image upload, meal plan import from JSON files, and dynamic form updates based on selected image models. All asynchronous operations are handled using native `async/await` patterns to ensure non-blocking behavior and responsive updates.

The image upload functionality supports both manual file selection and drag-and-drop interaction. Internally, a hidden file input element is programmatically triggered when the user clicks on the upload area. For drag-and-drop, event handlers capture the dropped file and convert it into a temporary object URL, which is stored in the application state for preview and later submission. This functionality ensures compatibility with common user interaction patterns while maintaining a minimal and reactive component structure.

6.2 Python Package

This section presents the implementation of the Python package developed as the core component of the system. It details the internal structure, main processing pipeline, and the methods used to perform the image generation.

6.2.1 Pipeline

The pipeline serves as the central entry point and output interface of the Python package. It takes a meal plan and an input image as input and returns a modified image reflecting the desired transformation. Internally, the pipeline comprises three main stages: parameter validation, prompt generation, and image generation.

In the first stage, the inputs are validated to ensure correctness and compatibility. This includes checking the format and content of the meal plan, as well as verifying that the image meets resolution and type requirements. Once validated, the prompt generation constructs a textual description (prompt) based on the input data, tailored to the requirements of the selected image model. Finally, the image generation uses this prompt to produce the edited image.

6.2.2 Parameter Validation

The first step of the pipeline involves validating the parameters received from external input. Certain parameters are mandatory for the pipeline to function correctly, while others are optional and only required under specific conditions. This validation step ensures that all necessary information is present and correctly formatted before further processing begins. All parameter requirements can be seen in the table 6.1

6.2.3 Prompt Generation

Before an image can be modified, a textual description of the intended transformation must be created. This description, referred to as the prompt, guides the image generation process and is tailored to the requirements of the chosen image model. To generate this prompt, the system uses one of two large language models: either the `Qwen3-30B` model via LLMHub or OpenAI's `GPT-4o`.

Parameter	Required	Description
image	Yes	Must be a Pillow(PIL) Image object.
meal plan	Yes	The meal plan itself.
language model	Yes	The language model to use. (OpenAI GPT-4o, LLMHub Qwen3-30B)
image model	Yes	The image model to use. (OpenAI GPT-Image-1, Null-text Inversion, ControlNet)
OpenAI api key	(No)	API key for OpenAI, used to instantiate the client. Required if model is chosen.
LLMHub api key	(No)	API key for LLM Hub, used to instantiate the client. Required if model is chosen.

Table 6.1: Parameter description and requirement

Prompt generation with Qwen3-30B The LLMHub provides us with the Qwen3-30B. We are mainly using it because of privacy and local-execution reasons. Compared to OpenAI's GPT-4o, it works similarly well for our use-case. The only deficit being that it has a average time of around 14 seconds for generation, which can also fluctuate. This was tested for exactly 5 requests directly to the Python package and only the prompt generation took us 14 seconds in average using the Qwen3-30B model.

Prompt Generation with GPT-4o We have decided to use GPT-4o, as it is one of the stronger models, which does not do reasoning. Reasoning can increase the response duration, which then adds more time to our whole pipeline, which we want to avoid if possible.

The prompt generation process is divided into two or three sequential stages, depending on the needs of the selected image model. First, the system checks whether the image model requires a caption of the original image. If so, a visual description must be generated prior to creating the transformation prompt. The second step, which is always required, involves interpreting the provided meal plan to extract relevant goals or attributes. In the final step, this information is used to construct a model-specific prompt, taking into account the prompting and expectations of the chosen image model.

Step 1: Caption Generation The prompt generation process begins with a conditional check to determine whether the selected image model requires a caption of the original image. This is particularly relevant for the NTI model, which relies on a dual-prompt-input approach: a base caption describing the original image and a separate prompt specifying the desired transformation. Consequently, if NTI is chosen for image generation, a textual description of the input image must first be created.

To generate such a caption, a visual model capable of describing both the person and the background in the image is required. However, LLMHub does not currently offer a model with image captioning capabilities. As a result, Null-text Inversion cannot be used in combination with LLMHub.

Step 2: Interpreting the Meal Plan For image models that do not require a base caption, the prompt generation process begins with the interpretation of the meal plan. In this step, the meal plan is submitted to the language model together with a carefully designed instruction prompt 1. The goal is to produce a concise textual summary that captures the expected visual changes in a person's appearance resulting from their dietary habits.

The generated output referred to as the *reflection in appearance* summarizes the likely physical effects based on the nutritional content and other user data provided in the meal plan. For example, a plan consisting of low activity levels, poor nutritional balance, and frequent unhealthy meals may yield a reflection such as:

> *Reflection in appearance: gains some weight, looks a little tired, and has less vibrant skin.*

This reflection forms the basis for the final prompt used in the subsequent image generation step.

Step 3: Constructing the Final Prompt The final step in the prompt generation process involves composing the actual prompt that will be used for image generation. This requires an additional request to the selected language model. Since different models expect different input formats and levels of detail, image model-specific instructions are provided to the LLM.

These instructions are implemented as predefined prompt templates, each associated with a corresponding image model via an extended enumeration structure. Once the image model is selected, the appropriate template is automatically retrievable and used to guide the LLM. Each template includes a clear instruction and examples to guide the LLM towards producing a prompt in the desired structure and tone, all prompts are listed in the Prompt Listings. A special case arises with the Null-text Inversion model, which builds upon the previously generated image caption. In this case, the final prompt must extend the original caption using the reflection in appearance obtained from the meal plan.

With these three steps completed, the final prompt is now ready to use in the image generation stage. This concludes the prompt generation component of the pipeline.

6.2.4 Image Generation

In this part, we will describe the last part of the pipeline, the image generation. We are using 3 different models, with which the images can be generated. We have integrated OpenAI's GPT-Image-1 model, which can be called using an API-key. Then we have two local models: ControlNet and Null-text Inversion.

OpenAI GPT-Image-1

The integration of the OpenAI GPT-Image-1 is simple. Using OpenAI's Python library, we can use the Image API to create an edit request. Before the actual call, we have to create

an image stream from the Pillow image, in order to send the image via API, along with the prompt, which we created in the prompt generation stage of the pipeline. The only parameter which is accepted, is the level of quality.

Null-text Inversion Integration

As for Null-text Inversion, we are using a pipeline from one of our advisors as the base. The whole pipeline works as follows:

1. Resizing the image to 512x512 so it fits the underlying Stable Diffusion model.
2. Reversing the diffusion process to approximate the original latent noise of the image using an inverse scheduler.
3. Do the Null-Text-Optimization
4. Generating new images from the inverted latents using classifier-free guidance, now optionally conditioned on a modified prompt.
5. Returning both the reconstructed base image and edited variations.

We made several key modifications to the base repository. One significant improvement was implementing a new way of image resizing. Previously, the image was cropped from the center, which posed a risk for images with extreme aspect ratios—either very vertical or very horizontal. This central cropping often removed substantial portions of the image, potentially excluding critical elements such as faces or other important features. To solve this issue, we added padding to the image, so that it is a square. So instead of zooming and cropping, we added padding to the uneven sides.

Another major issue we addressed was the inefficient model loading process. In some cases, memory was not properly released between requests, resulting in severe performance degradation. Image generation times stacked up with ongoing requests, and sometimes exceeded 10 minutes, which was a critical error for us. We had two solutions for this problem. First of all, we emptied the GPU cache after every generation. Secondly, we modified the system to bundle the model directly within the package. While the model increases the overall package size, it offers a trade-off by providing more consistent and predictable image generation times. Previously, relying on a cached model led to significant variability in performance. Both of these changes together helped us create a consistent performance.

ControlNet Integration

To integrate ControlNet into our system, we utilized the Python package Auto1111SDK, which is derived from the same codebase as the Stable Diffusion Web UI. While the full Web UI is overly complex and resource-intensive for our use case, Auto1111SDK offered a lighter and more modular alternative, making it suitable for embedding ControlNet functionality directly into our Python package. Our motivation for adopting a solution similar to the Stable Diffusion Web UI lies in its comprehensive suite of tools for image editing. It provides a wide

range of adjustable parameters and settings, enabling fine-grained control over the image generation process.

Limitations of Auto1111SDK Although Auto1111SDK initially appeared to be a suitable solution, it introduced several significant limitations. First, the package has not been actively maintained for over a year. The most recent update—ControlNet integration—was added only in an unstable release. While we were fortunate that this feature was included, its implementation is not yet robust. For instance, we were unable to modify all default parameters through the standard model pipeline interface and had to resort to injecting configurations directly into the class to achieve the desired behavior.

Additionally, Auto1111SDK lacks support for in-memory image inputs, requiring all images to be written to disk before processing. This introduced unnecessary I/O overhead and limited the flexibility of our system. To address this, we also had to manually inject additional functionality into the class. As we do not want to save any images persistently in any way.

Despite these drawbacks, Auto1111SDK remains the most practical option currently available for approximating the functionality of the Stable Diffusion Web UI.

6.2.5 Package Configuration

The Python package is structured using a standard `pyproject.toml` configuration, and a `setup.py` with source files organized under a `src/` directory and dependencies declared explicitly. Installation requirements include commonly used libraries for image processing and model integration, such as `Pillow`, `diffusers`, and `openai`. Due to compatibility constraints, the version of `diffusers` is fixed at 0.29.0, as newer versions require `transformers` versions that are incompatible with the AutoSDK1111 library. Furthermore, both `torch` and the `Auto1111SDK` have to be installed manually.The installation guide for the Python package can be viewed here: A.1.

6.2.6 Built package

Python mainly has .whl files and .tar.gz-files as distribution formats. Python `.whl` files, also known as *wheel files*, are the standard built-package format for distributing Python packages. Unlike source distributions (e.g., `.tar.gz`), wheel files are precompiled, allowing for faster and more reliable installations. Our Python package has both of these, but we do prefer to use the wheel file and have also used the .whl-file in the installation guide.

These are key features of wheel files:

- **Binary Distribution:** Wheel files contain built distributions, avoiding the need to compile code (such as C extensions) during installation.
- **Efficient Installation:** Since no build step is required, installing from a wheel is significantly faster than from a source distribution.

- **Platform-Specific:** Some wheels are platform- and Python-version-specific if they include compiled components.

6.3 API

This section describes the implementation of the API responsible for handling controlled image generation requests. The API is implemented in Python using the `FastAPI` framework.

The API serves as the interface between the client (e.g., the web frontend) and the internal controlled image generation pipeline. It handles file uploads, parameter collection, task dispatching, and asynchronous result delivery. Internally, all requests are managed through a queue-based job processing system to ensure reliable and isolated execution of resource-intensive tasks.

6.3.1 FastAPI

The API is implemented using FastAPI, a modern web framework that supports asynchronous request handling, and automatic OpenAPI (Swagger) documentation. Additionally, we segregated each endpoint so that it has one responsibility. But, we still have one generate endpoint for all combinations for testing purposes.

The backend exposes a modular set of REST endpoints grouped by LLM backend and image generation model:

- `POST /generate/`: Unified endpoint that supports all combinations of LLMs and image models through parameterized input.
- `POST /generate/openai/openai`: Uses OpenAI for both prompt and image generation.
- `POST /generate/openai/nto`: Uses OpenAI for prompt generation and Null-Text Optimization (NTO) for image transformation.
- `POST /generate/openai/controlnet`: Uses OpenAI with ControlNet as the image model.
- `POST /generate/llmhub/openai`: Uses LLMHub for prompt generation and OpenAI for image editing.
- `POST /generate/llmhub/controlnet`: Uses LLMHub for prompt generation and ControlNet for image generation.
- `GET /status/{task_id}`: Returns job status and progress information.
- `GET /result/{task_id}`: Returns the generated image (base64-encoded PNG) and the textual reflection prompt.

All endpoints receive input as multipart form data, consisting of an image file, a JSON-encoded meal plan, API keys, and model configuration parameters. A shared `JobQueue` instance is injected into each route using FastAPI's `Depends()` pattern, ensuring a clean separation of concerns and access to globally tracked job states.

Meal Plan The meal plan represents the user's data extracted from the Smart Eating Platform. To ensure consistency and validity, a dedicated JSON schema **??** was developed. The final schema includes the following key components:

1. Anonymized user data
2. A detailed listing of meals representing the diet plan
3. Nutritional information, like the macronutrient and micronutrient values

6.3.2 Concurrency

Due to the computational intensity of image generation, particularly when using diffusion-based models or large language models, the API is designed to decouple request handling from image processing through an asynchronous job queue.

Each incoming request is encapsulated as a Job object, which holds all relevant input data and metadata. Jobs are placed into an `asyncio.Queue`, from which they are consumed by a dedicated background worker. This worker executes the image generation process sequentially in a separate thread using `run_in_executor()`, ensuring that the FastAPI event loop remains non-blocking.

Jobs track their own processing state and status flags. After completion, results are stored temporarily in memory and can be retrieved through the `/result/{task_id}` endpoint. To manage memory usage, the system explicitly releases resources via Python's garbage collector (`gc.collect()`) following each generation along with the `del` keyword to dereference an object.

To prevent GPU overload or memory contention, the system is currently configured to process only one job at a time (`_max_concurrent_jobs = 1`). However, this default setting can be increased to use parallelization by scaling the number of concurrent worker tasks.

Chapter 7

Evaluation

This chapter focuses on the user-centered evaluation of image generation quality using a structured Likert-scale rating procedure. While prior testing identified technically optimal parameters and model behaviors, this evaluation captures how the generated images are perceived by human raters in terms of visual plausibility, identity preservation, and alignment with the described transformation.

7.1 User Image Rating

To assess subjective image quality, we conducted a complementary evaluation using a 5-point Likert scale. This approach allowed raters to express how well each image matched the described physical transformation and preserved the subject's identity.

7.1.1 Evaluation Procedure

Participants were presented with 18 images, six from each of the three generation pipelines:

- GPT-4o,
- Null-text Inversion,
- ControlNet.

Each image was shown alongside the text prompt describing the intended transformation. Raters were asked to answer the following question:

> *"The generated image accurately reflects the appearance changes described in the text."*

Responses were collected using the Likert scale shown in Table 7.1.

Rating Option	Score
Strongly agree	+2
Agree	+1
Neutral	0
Disagree	–1
Strongly disagree	–2

Table 7.1: Likert Scale Options and Score Encodings

A total of nine raters completed the form, resulting in 162 individual evaluations.

7.1.2 Aggregated Results

To analyze the distribution of ratings across the models, we visualized the results using a boxplot (Figure 7.1). Each box represents the interquartile range (25th to 75th percentile), with the horizontal line indicating the median and the cross (×) marking the mean. Whiskers extend to the minimum and maximum values within 1.5 times the interquartile range.

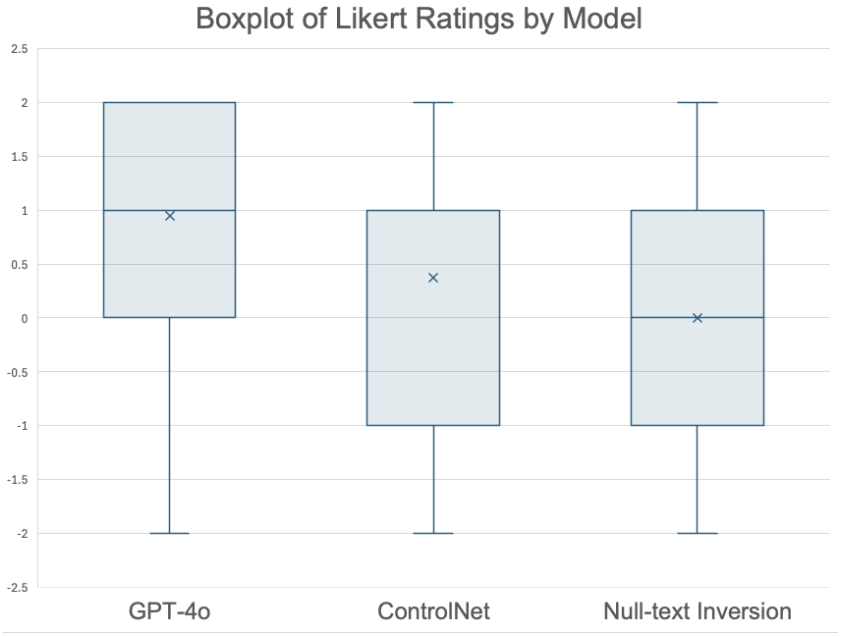


Figure 7.1: Boxplot of Likert Ratings by Model

The results show that GPT-4o achieved the highest ratings overall, with a median of 1 and a mean rating of 0.94. This indicates that most raters agreed or strongly agreed that GPT-4o's outputs accurately reflected the described transformations while maintaining the subject's identity. The relatively narrow interquartile range and low variance suggest a high level of consistency across different prompts and input images.

ControlNet outperformed Null-text Inversion in terms of average rating, with a mean of 0.37. Its median was 1, and the ratings ranged across the full Likert spectrum from –2 to +2. The broad distribution implies that while some outputs were well-received, others failed to meet expectations. This mixed performance reflects the model's sensitivity to prompt quality and parameter tuning, consistent with earlier findings in our exploration and Elo-based evaluations.

Null-text Inversion received the lowest mean rating, at 0.00, and a median of 0. The interquartile range spans from –1 to 1, indicating that evaluations were balanced between mildly positive and mildly negative. This suggests a high degree of inconsistency: while some results were acceptable, many failed to convincingly reflect the intended changes or preserve identity. Given its theoretical strength in semantic editing, the practical shortcomings observed here may reflect suboptimal parameter selection for the specific image, image resolution constraints, or the general challenge of text-conditioned inversion.

7.1.3 Conclusion

The boxplot analysis complements and reinforces the Elo-based evaluation. GPT-4o demonstrated the strongest performance both in rating level and consistency, making it the most reliable option from a user perception standpoint. ControlNet, although less consistent, showed a higher average score than Null-text Inversion, indicating greater practical effectiveness in this specific use case. These results validate the decision to offer GPT-4o as a fallback model while retaining the open-source pipelines for local, privacy-sensitive, or customizable deployments.

7.2 Qualitative Results

While the Likert-scale evaluation provided a structured and quantitative assessment of user-perceived image quality, it is equally important to examine the outputs from a qualitative perspective. This section presents representative examples from each generation pipeline that were used in the user study described above. These examples illustrate the visual strengths and limitations of each model in terms of identity preservation, realism, and alignment with the described transformation.

Each image was paired with a reflection in appearance describing the intended physical change. The following subsections provide a model-wise overview, showcasing selected samples and discussing the typical visual behavior observed across the test set.

7.2.1 ControlNet

ControlNet produced some of the most visually detailed and realistic outputs overall. In many cases, facial features, textures, and lighting were rendered with a high degree of photorealism, resulting in images that felt natural and credible (see Figure 7.2). This strong baseline realism made ControlNet outputs appear more grounded than those from GPT-4o.

However, the model showed inconsistent responsiveness to prompts. Depending on the wording, some images reflected the intended transformation very clearly, while others showed little to no change from the input (see Figure 7.3).

Reflection in appearance: Gains some muscles, more vibrant skin, lean and athletic build.

Figure 7.2: ControlNet example: Realistic transformation with subtle changes. Original photo by Timothy Barlin from Unsplash.

Reflection in appearance: gains some weight, looks more tired.

Figure 7.3: ControlNet example: Realism is high, but prompt adherence is not given. Original photo by Reza Biazar from Unsplash.

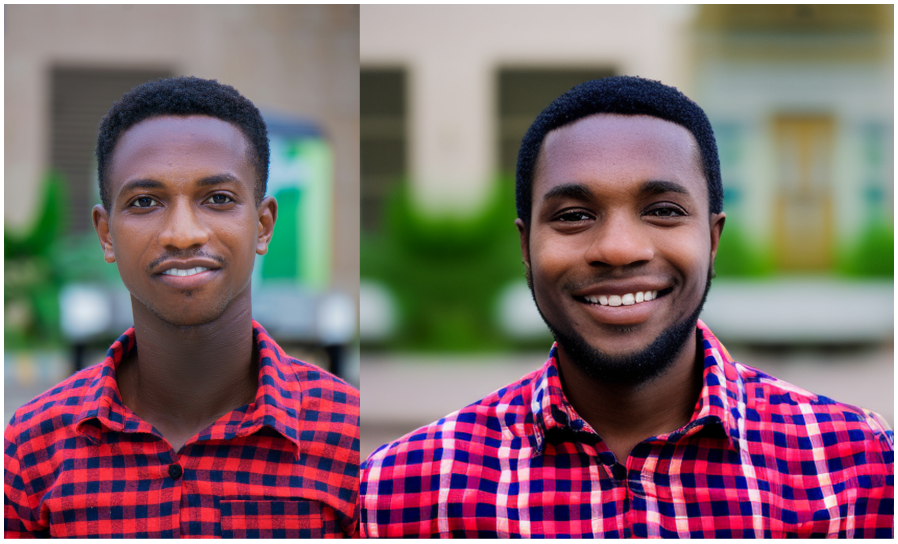
Null-text Inversion

Null-text Inversion offered more visibly prominent transformations when it succeeded, often altering body or facial features in a way that clearly reflected the prompt. Its outputs tended to be slightly less detailed than those of ControlNet but were generally photorealistic and semantically aligned with the intended change (see Figure 7.4).

That said, consistency was a limitation. Some outputs changed the subject's identity significantly, such as adding facial hair or altering bone structure, which undermined the realism and coherence of the transformation (see Figure 7.5).

Reflection in appearance: gains weight, looks more tired

Figure 7.4: Null-text Inversion example: Noticeable transformation with good identity preservation. Original photo by Reza Biazar from Unsplash.



Reflection in appearance: Likely lean and toned with muscular definition, more vibrant skin, healthy weight.

Figure 7.5: Null-text Inversion example: Transformation is strong, but identity preservation is poor. Original photo by Ransford Quaye from Unsplash.

GPT-4o

GPT-4o generated images that were technically impressive, with smooth, artifact-free textures and high fidelity to the prompt (see Figure 7.6. The model excelled at prompt adherence as virtually every output reflected the described transformation in a clear and unambiguous way.

However, this strength also introduced a tendency toward exaggeration. In several cases, the model over-interpreted relatively mild prompts. For instance, a request to “make him gain some weight” resulted in extreme obesity, with the subject appearing nearly unrecognizable or comical in proportion (see Figure 7.7). Moreover, some images had an “AI-generated” appearance, overly perfect textures, artificial symmetry, or lighting that made them feel less natural than ControlNet’s or Null-text Inversion’s outputs.

Reflection in appearance: Gains lean muscle, more vibrant skin, appears fit and energetic.

Figure 7.6: GPT-4o example: Clear prompt adherence and high resemblance, not overly exaggerated. Original photo by Ransford Quaye from Unsplash.

Reflection in appearance: gains weight, looks more tired.

Figure 7.7: GPT-4o example: Prompt adherence is strong, but the result is exaggerated. Original photo by Jurica Koletić from Unsplash.

Summary of Observations

The following qualitative patterns emerged from the user evaluation examples:

- **ControlNet** delivered the most natural and photorealistic results but was less responsive to subtle prompts. When changes did occur, they were often well-integrated and realistic.
- **Null-text Inversion** provided stronger transformations than ControlNet but was inconsistent in preserving subject identity. Realism was acceptable, but less refined in detail.
- **GPT-4o** offered unmatched prompt adherence and visual clarity but often exaggerated the requested transformation. Some outputs lacked the nuanced realism of the other models, instead showing a synthetic or AI-generated feel.

These examples support the earlier quantitative findings and highlight the trade-offs between realism, controllability, and prompt sensitivity inherent to each approach.

Chapter 8

Conclusion

With this chapter, we conclude our thesis. We begin by elaborating on the results we have achieved, followed by a discussion of the improvements derived through the course of this work.

8.1 Results

This thesis explored the integration of controlled image generation into a personalized nutrition platform. Our core contribution is the development and evaluation of a system that visualizes the physical outcomes of a dietary plan by transforming user images based on AI-generated predictions. This approach combines language and image models to provide users with intuitive, personalized, and motivational visual feedback.

This section summarizes the key findings from our experimental and user-based evaluations.

8.1.1 Evaluation

The primary result of this thesis is the successful application of image models to modify avatars in a manner that reflects a person's dietary habits. Through comprehensive evaluation, we assessed the performance and effectiveness of several models in achieving this goal.

Our focus centered on two open-source models: ControlNet and Null-text Inversion. While we initially considered using InstructPix2Pix, early experimentation revealed it to be unsuitable for our requirements. In contrast, ControlNet and Null-text Inversion showed considerable promise. We subsequently adopted a systematic evaluation methodology, generating several hundred images using a defined set of base images, weights, and prompts.

To determine the optimal weight configuration, we compared images derived from identical base inputs and prompts. By employing an Elo rating system to rank these transformations,

we were able to quantitatively distinguish between weight sets and identify the most effective configuration for each model.

The Likert-scale evaluation provided strong evidence for the relative strengths and weaknesses of each model. GPT-4o achieved the highest ratings overall, with a mean score of 0.94 and a median of 1, indicating that most raters agreed or strongly agreed that its outputs reflected the target transformations while maintaining visual realism and identity. The model's narrow interquartile range and low variance further suggest consistent performance across different inputs and prompts.

ControlNet followed with a mean score of 0.37. While its median rating was also 1, indicating that several outputs were positively received, the wider spread across the full Likert scale—from –2 to +2—points to mixed quality. This variability is consistent with the findings from our earlier Elo-based analysis and reflects the model's sensitivity to parameter tuning and prompt formulation.

Null-text Inversion scored lowest, with a mean rating of 0.00 and a median of 0. The interquartile range extended from –1 to +1, suggesting a more neutral perception overall. This indicates a significant inconsistency in performance: although some images adhered well to the transformation prompt, many did not convincingly maintain identity or semantic relevance. These results align with our earlier qualitative findings that Null-text Inversion often produces prompt-aligned outputs at the expense of identity preservation and structural fidelity.

Midway through our project, OpenAI released a proprietary image model which significantly exceeded our expectations. This model consistently produced high-quality outputs with strong prompt adherence while preserving the identity of the subject, although some minor inconsistencies in background fidelity were noted.

8.1.2 Software Deliverables

Among our deliverables is a custom Python package developed to facilitate image editing within the Smart Eating Platform. To enhance testing and usability, we implemented a user interface and integrated the package with a FastAPI backend. This backend queues incoming requests and processes them sequentially to maintain performance consistency.

Each endpoint corresponds to a specific combination of image and language models, adhering to the *Interface Segregation Principle* from the SOLID principles. This design choice ensures modularity and facilitates maintainability.

The Python package itself is composed of several distinct stages:

- **Parameter Validation**: Ensures input data is complete and consistent.
- **Prompt Generation**: Constructs a "reflection in appearance" from a user's meal plan. This intermediate form translates the dietary input into a model-specific visual transformation prompt.

- **Image Generation**: Utilizes the generated prompt to transform the input image using the selected model.

The final output includes both the generated image and the associated reflection in appearance, providing a cohesive and interpretable representation of the transformation.

All of our functional requirements, which we have defined in 3.2 have been met. But the non-functional requirements have not fully been met. There are two main exceptions here, which we could not meet. The first one being the limitation of Python version 3.10. Due to the dependency on the Python library Auto1111SDK, we are limited to Python 3.10 and do not support newer Python versions.

8.2 Future Outlook

This section outlines potential improvements and extensions to our current implementation. Several features could not be included due to time constraints, but they represent promising directions for future development.

One key enhancement would be the integration of Null-text Inversion with LLMHub. Currently, LLMHub does not support this feature because it requires a textual caption derived from the base image. Unfortunately, the Qwen3 language model used in our project lacks image captioning capabilities. An alternative approach would have involved using an open-source visual model capable of generating image descriptions, thereby enabling the use of Null-text Inversion.

Another significant improvement would be the adoption of websockets in place of traditional API endpoints for backend communication. During the latter stages of the project, we realized that image generation was more time-consuming than initially anticipated. Our current backend processes requests sequentially in a queue, which introduces latency. Implementing a websocket-based architecture could enable real-time communication and status updates. However, this approach was not pursued due to our limited experience with websockets and the late-stage nature of this realization.

A final area for enhancement involves conducting a more comprehensive evaluation of alternative image generation models. Due to time limitations and the need to maintain focus, we restricted our exploration to a narrow subset of models. With more time, broader experimentation could have led to improved results and more informed model selection.

Beyond the immediate application within the Smart Eating Platform, this thesis illustrates the growing feasibility of integrating generative AI into health and wellness tools. As generative models become more accessible and customizable, we expect systems like ours to play a larger role in behavior-driven applications, offering users more intuitive, emotionally resonant feedback that traditional metrics alone cannot provide.

In summary, while our implementation achieved its core objectives, there are numerous opportunities to improve and expand the system. Future work could address these limitations

to enhance performance, usability, and scalability.

Appendix A

Technical Documentation

A.1 Installation Guide

The following section is the installation guide for the Python package, to be able to use the API. Download and install a Python 3.10.x from python.org. Python version 3.10.11 is the latest version, that has an installer.

The following commands create and activate a new Python virtual environment:

```
py -3.10 -m venv venv_cig
.\venv\_cig\Scripts\activate.ps1
```

Update pip before installing any packages:

```
pip install pip --upgrade
```

A.1.1 Torch and CUDA

Torch is a library from Meta, dedicated to AI. CUDA is a library from NVIDIA that allows code to interface with NVIDIA GPU hardware.

Note: Simple Torch installations do not use CUDA. For CUDA support, Torch must be compiled with the appropriate CUDA libraries. Likewise, `torchvision` has CUDA-enabled packages.

System-level CUDA drivers are managed via tools like `nvidia-smi.exe`, which shows the installed CUDA version. As of June 2025, our system-level CUDA version is 12.2.

To ensure compatibility and avoid conflicts, we install a specific CUDA runtime within the Python environment:

```
python -m pip install --verbose nvidia-cuda-runtime-cu11
```

Install CUDA-enabled Torch and torchvision:

```
python -m pip install --verbose torch==2.1.0 torchvision --index-url https://download.pyt
```

Validation

Verify the installation of Torch and CUDA:

```
py
>>> import torch
>>> print("Torch version:", torch.__version__)
>>> print("Is CUDA enabled?", torch.cuda.is_available())
```

Expected output:

```
Torch version: 2.1.0+cu118
Is CUDA enabled? True
```

A.1.2 Install Torch-dependent Package

Now install a Torch-based dependency:

```
pip install git+https://github.com/saketh12/Auto1111SDK.git
```

A.1.3 Install the Wheel Package

Install the prebuilt wheel file, you can find the file inside of the python_package folder under the /dist folder:

```
pip install .\controlled_image_generation_ost-0.1.0-py3-none-any.whl
```

Ensure the path to the `.whl` file is correctly specified.

Acknowledgment

Part of this guide references material from the Auto1111SDK GitHub repository: Auto1111SDK Installation Guide.

A.2 Setup Python package development environment

Before being able to develop, the full installation guide must be done.

To setup a local development environment, we need the the full package structure findable in the final gitlab repo. When inside the package directory, activate your virtual environment, if needed and use the command:

```
With virtual environment:
pip install -e

Without virtual environment:
py -3.10 pip install -e
```

This command links your package to the virtual environments libraries. This way you can use the Python package dynamically, without having to install the build file to your system after each change.

To build the package use the following command, when you are inside of the Python package folder (if you have a virtual environment activate it before using the command):

```
with virtual environemnt:
build
without virtual environment
py -3.10 -m build
```

A.3 Test Summary

This section shows and covers all tests that we have concluded using our API and our Python package. Our tests mainly evolve around the use of our API, which in turn uses the Python package. The tests have been written using pytest with FastAPI and are black-box tests derived from functional requirements

Test ID	Test Description	Status
T1	Test generate API without segregation using OpenAI as language and image model	Finished & Passed
T2	Test generate API for OpenAI and Null-text Inversion	Finished & Passed
T3	Test generate API for OpenAI and ControlNet	Finished & Passed
T4	Test generate API for OpenAI and OpenAI	Finished & Passed
T5	Test generate API for LLMHub and OpenAI	Finished & Passed
T6	Test generate API for LLMHub and ControlNet	Finished & Passed
T7	Handling 18 generation requests, while generating in the background.	Finished & Passed
T8	Test failing generate API for LLMHub and Null-text Inversion	Finished & Passed
T9	Failing test if meal plan is empty	Finished & Passed
T10	Failing test, try random task id pulling	Finished & Passed
T11	Failing test, try pulling job status, after fetching result.	Finished & Passed

Table A.1: Test cases in API with Python package

A.4 Requirement Protocol

The requirement protocol lists all requirements defined in the chapter 3 consisting of functional and non-functional requirements.

A.4.1 Functional Requirements

This subsections shows the acceptance list A.4.1 for all functional requirements.

FR ID	Description	Acceptance Criteria	Test Method	Status
1.1	UI must accept a meal plan.	Does accept meal plan.	Manual	Done
1.2	UI must accept a user's picture (image file).	Does accept image	Manual	Done

Continued on next page

Table A.2 continued from previous page

FR ID	Description	Acceptance Criteria	Test Method	Status
1.3	UI must trigger generation using parameters and show result.	Does trigger with inputs, concluded by output.	Manual	Done
1.4	UI must allow preview and download of modified images.	Result is downloadable	Manual	Done
1.5	UI must not allow Null-text inversion and LLMHub in combination.	Null-text inversion is not visible when LLMHub is chosen and if it is already chosen, automatically switches to OpenAI as soon as LLMHub is chosen while Null-text Inversion is selected.	Manual	Done
2.1	Package must summarize the meal plan into key sentences.	Key sentences are generated using the meal plan and the summarization prompt.	Manual / Derived from test results and prints	Done
2.2	Package must use summary to create image model specific prompts.	Image model specific prompts are created.	Manual/ Derived from test results and prints that prompts are model specific.	Done
2.3	Package must use the selected LLM.	The selected LLM is used in the package.	Manual / Derived from test prints	Done
3.1	System must use AI-based image processing to modify user's profile picture.	Yes, the system is using AI-based image processing.	-	Done
3.2	System must use the selected image model by the user.	Yes, the system uses the selected image model.	Derived from test results and prints.	Done

Continued on next page

Table A.2 continued from previous page

FR ID	Description	Acceptance Criteria	Test Method	Status
4.1	Image generation pipeline implemented as a Python package.	The whole pipeline to generate one image using a meal plan was implemented as a Python package	Manual	Done
4.2	Package takes in and validates inputs.	Derived from positive and negative meal plan test results that it is validated.	Derived from test results.	Done
4.3	Package returns edited image and reflection in appearance.	The package does return both, the edited image and a reflection in appearance.	Derived from test results.	Done
5.1	API provides endpoints to integrate into other platforms.	The API provides endpoints, that can be used.	Derived from test results.	Done
5.2	Endpoints segregated using Interface Segregation Principle.	The endpoints are segregated into 5 different endpoints, with no unnecessary API parameters.	Derived from test results	Done
5.3	API handles multiple requests using a queue and processes sequentially.	Yes, multiple requests can be sent over all endpoints. They are enqueued and worked off sequentially.	Derived from test results T7	Done

Table A.2: Functional Requirements with Acceptance Criteria and Test Columns

A.4.2 Non-Functional Requirements

This subsections shows the acceptance list A.4.1 for all non-functional requirements.

ID	Description	Acceptance Criteria	Test Method	Status
NFR 1.1	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 25s) using all model combinations. Calculate average over 5 different image generations.	Pipeline duration measured and within landing zone thresholds.	Defined method for all NFR 1.1 iterations.	Done
NFR 1.1a	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 25s) using OpenAI's LLM and OpenAI's Image model.	Pipeline duration measured and within landing zone thresholds.	Well within regular at around average 35 seconds.	Done
NFR 1.1b	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 20s) using OpenAI's LLM and ControlNet	Pipeline duration measured and within landing zone thresholds.	Within regular with average around 50 seconds.	Done
NFR 1.1c	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 20s) using OpenAI's LLM and Null-text Inversion	Pipeline duration measured and within landing zone thresholds.	Within the minimum landing zone, close to regular with average around 66 seconds. Depends a lot on image with big variance between images.	Done
NFR 1.1d	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 25s) using LLMHub's Qwen3 and OpenAI's image model.	Pipeline duration measured and within landing zone thresholds.	Within regular with average around 50 seconds.	Done
NFR 1.1e	Full image generation completes within defined landing zones for pipeline duration (min 2m, reg 1m, out 20s) using LLMHub's Qwen3 and ControlNet.	Pipeline duration measured and within landing zone thresholds.	Around 64 seconds average within minimum zone and close to regular zone.	Done

Continued on next page

Table A.3 continued from previous page

ID	Description	Acceptance Criteria	Test Method	Status
NFR 1.2	Prompt generation subprocess completes within defined landing zones (min 10s, reg 3s, out 1s) for an average (30 day) meal plan. Calculate average over 5 requests.	Prompt generation time measured and within landing zones.	Define Method for all NFR1.2	Done
NFR 1.2a	Prompt generation subprocess completes within defined landing zones (min 10s, reg 3s, out 1s) for average meal plan using OpenAI.	Prompt generation time measured and within landing zones.	Close to regular within minimum with average around 4 seconds.	Done
NFR 1.2b	Prompt generation subprocess completes within defined landing zones (min 10s, reg 3s, out 1s) for average meal plan using LLMHubs Qwen3.	Prompt generation time measured and within landing zones.	Not within landing zones with an average around 14.5 seconds.	Failed
NFR 1.3	Image transformation step completes within landing zones (min 2m, reg 40s, out 20s) for all integrated models (OpenAI, ControlNet, Null-text Inversion).	Transformation duration per model measured and within landing zones.	Defines method for all NFR 1.3	Done
NFR 1.3a	Image transformation step completes within landing zones (min 2m, reg 40s, out 20s) for OpenAI.	Transformation duration per model measured and within landing zones.	Withing regular zone with average around 32 seconds.	Done
NFR 1.3b	Image transformation step completes within landing zones (min 2m, reg 40s, out 20s) for ControlNet.	Transformation duration per model measured and within landing zones.	Withing minimum zone with average around 48 seconds, close to regular zone.	Done
NFR 1.3c	Image transformation step completes within landing zones (min 2m, reg 40s, out 20s) for Null-text Inversion.	Transformation duration per model measured and within landing zones.	Withing minimum zone with average around 67 seconds.	Done

Continued on next page

Table A.3 continued from previous page

ID	Description	Acceptance Criteria	Test Method	Status
NFR 2.1	AI-generated profile pictures are realistic and visually plausible, without extreme distortions.	Visual inspection and user feedback confirm realism.	Done with Evaluation forms.	Done
NFR 2.2	System prevents exaggerated transformations that could mislead users.	No extreme or misleading transformations observed in outputs.	Done with Evaluation Forms.	Done
NFR 2.3	Transformations reflect weight changes, muscle growth, athleticism, tiredness according to meal plan.	Generated images show expected visual cues per meal plan.	Done with Evaluation Forms.	Done
NFR 3.1	Uploaded images are processed in-memory only; no persistent storage of images.	No image files saved on disk or persistent storage during processing.	Manually checked and implemented in a way that nothing is saved persistently.	Done
NFR 4.1	System is modular and allows easy extensibility.	Codebase structured in modules with documented extension points.	Code architecture review and developer feedback.	
NFR 5.1	UI is simple and intuitive; users can upload images and meal plans easily; generation starts with one click; results shown without extra effort.	User testing confirms ease of use and straightforward workflow.	Testing with user new to the system.	Done
NFR 5.2	Web UI is responsive and works well on desktop and mobile devices.	UI layout adapts correctly on various screen sizes and devices.	Responsiveness tested manually.	Done
NFR 6.1	Python package compatible with Python 3.9 and newer.	Package installs and runs without errors on Python 3.9+.	Due to dependencies we are limited to only Python 3.10.x	Failed
NFR 6.2	UI supports major browsers: Chrome, Firefox, Edge, Safari.	UI functions correctly on all listed browsers.	Cross-browser compatibility tested.	Done

Continued on next page

Table A.3 continued from previous page

ID	Description	Acceptance Criteria	Test Method	Status

Table A.3: Acceptance Checklist for Non-Functional Requirements

A.5 Prompt Listings

```
text_generation_prompt = (
    "You get a meal plan from a person and predict how their nutrition could
    ↪ reflect on their appearance. Focus only on appearance changes - no
    ↪ comments about food/meal plans. Keep responses very short. Avoid
    ↪ details about skin beyond 'vibrant' (use examples below). Consider
    ↪ per-meal nutrient averages. Examples: \n\n30 days Meal Plan: Very bad
    ↪ food, poor nutrients, high calories, low activity → Reflection: gains
    ↪ little weight, looks tired \n\n120 days Meal Plan: Poor food, low
    ↪ activity → Reflection: gains some weight, tired appearance \n\n120 days
    ↪ Meal Plan: Protein-rich, good nutrients, high activity → Reflection:
    ↪ muscle gain, vibrant skin, lean \n\n120 days Meal Plan: Low calories,
    ↪ small portions, medium activity → Reflection: weight loss, vibrant
    ↪ skin, skinny"
)
```

Listing 1: Text generation prompt for appearance reflection based on meal plans

```
chatgpt_prompt = (
    "You get information on how a person has changed after following a certain
    ↪ lifestyle called 'Reflection in appearance', create a concise
    ↪ description of how the persons has changed in one precise sentence. We
    ↪ want to use it as a prompt for an image model, so follow the given
    ↪ example.  Example: Reflection in appearance: Her athletic and lean
    ↪ physique reflects her commitment to a meal plan that's well-structured
    ↪ and varied, focusing on balanced nutrition for a high activity level
    ↪ and a vibrant skin. The person looks very healthy. new prompt: Make the
    ↪ person look lean and athletic with a vibrant skin."
)
```

Listing 2: OpenAI GPT4-o prompt for generating final prompt used in image generation

```
controlnet_prompt = ("You get information on how a person has changed after
↪  following a certain lifestyle called 'Reflection in appearance', create a
↪  concise description of how the persons has changed in one precise sentence.
↪  We want to use it as a prompt for an image model, so follow the given
↪  example. Try to use the keywords as in these examples: Reflection in
↪  appearance: Her athletic and lean physique reflects her commitment to a
↪  meal plan that's well-structured and varied, focusing on balanced nutrition
↪  for a high activity level and a vibrant skin. The person looks very
↪  healthy. new prompt: Make the person look lean and athletic with a vibrant
↪  skin. Reflection in appearance:gains weight, looks more tired. new prompt:
↪  Make the person look obese and looking tired Reflection in appearance:
↪  gains significantly more weight, less vibrant skin new prompt: Make the
↪  person look extremely obese and less vibrant skin."
)
```

Listing 3: ControlNet prompt for generating for appearance reflection based on meal plans

```
null_text_inversion_prompt = ( "Using the base prompt given about a person,
↪  replace or extend the sentence using the changes reflecting their
↪  appearance. Place the new appearance as an adjective in front of the
↪  subject: man will become, slightly chubby man. black woman will become
↪  athletic and lean black woman. Do not change the base sentence too much.
↪  Don't enlarge the sentence too much, as it does not work with longer
↪  sentences. For example: base prompt: A black woman standing in a library in
↪  front of windows. new prompt: A lean and athletic black woman standing in a
↪  library in front of windows."
)
```

Listing 4: Null-text Inversion prompt for generating for appearance reflection based on meal plans

A.6 Used Tools

In this subsection we list all tools, which were used in this project:

Use-case	**Used tools**
Text generation, Text editing, Text improvement,	ChatGPT, Deepl Write, Write-full, Overleaf AI
Coding, Code generation, code improvement, code editing Idea generation Literature research	VSCode, Pycharm, ChatGPT, PerplexityAI, Claude ChatGPT, PerplexityAI, Claude Citavi, Arxiv, Google scholar, PerplexityAI, ChatGPT
Translation	ChatGPT, DeepL
Documentation	LaTeX, Overleaf
Communication tools	Outlook, Teams, GitLab, Jira
Data analysis and visualization	Excel, ChatGPT

Table A.4: Used tools

Controlled Image Generation API 0.1.0 OAS3

/openapi.json

default

POST `/generate/llmhub/controlnet` Generate Image With Llmhub Controlnet

Parameters Try it out

No parameters

Request body required multipart/form-data

image * required
`string($binary)` Input image file (PNG or JPG)

mealplan * required
`string` Meal plan in JSON string format

llmhub_api_key * required
`string` LLMHub API key

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** Schema `"string"`	*No links*
422	Validation Error Media type: application/json **Example Value** Schema `{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`	*No links*

POST `/generate/llmhub/openai` Generate Image Wwith Llmhub Openai

Parameters Try it out

No parameters

Request body required multipart/form-data

image * required
`string($binary)`
Input image file (PNG or JPG)

mealplan * required
`string`
Meal plan in JSON string format

llmhub_api_key * required
`string`
LLMHub API key

openai_api_key * required
`string`
OpenAI API key

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls `Accept` header. **Example Value** Schema `"string"`	*No links*
422	Validation Error Media type application/json **Example Value** Schema `{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`	*No links*

POST `/generate/openai/controlnet` Generate Image With Openai Controlnet

Parameters Try it out

No parameters

Request body required multipart/form-data

image * required
`string($binary)`
Input image file (PNG or JPG)

mealplan * required
`string`
Meal plan in JSON string format

openai_api_key * required
`string`
OpenAI API key

Responses

Code	Description	Links
200	Successful Response	*No links*

Code	Description	Links

Media type

application/json

Controls Accept header.

Example Value Schema

```
"string"
```

422 — Validation Error — *No links*

Media type

application/json

Example Value Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

POST /generate/openai/nto Generate Image With Openai Nto

Parameters Try it out

No parameters

Request body required — multipart/form-data

image * required
`string($binary)` — Input image file (PNG or JPG)

mealplan * required
`string` — Meal plan in JSON string format

openai_api_key * required
`string` — OpenAI API key

Responses

Code	Description	Links

200 — Successful Response — *No links*

Media type

application/json

Controls Accept header.

Example Value Schema

```
"string"
```

422 — Validation Error — *No links*

Media type

application/json

Example Value Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
```

Code	Description	Links


```
        ],
        "msg": "string",
        "type": "string"
      }
    ]
}
```


POST `/generate/openai/openai` Generate Image With Openai Openai

Parameters Try it out

No parameters

Request body required — multipart/form-data

image * required
`string($binary)` — Input image file (PNG or JPG)

mealplan * required
`string` — Meal plan in JSON string format

openai_api_key * required
`string` — OpenAI API key

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** Schema	*No links*
422	Validation Error Media type: application/json **Example Value** Schema	*No links*


```
"string"
```

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


POST `/generate/` Generate Image

Parameters Try it out

No parameters

Request body required — multipart/form-data

image * required
`string($binary)` — Input image file (PNG or JPG)

mealplan * required
`string` — Meal plan in JSON string format

llm_mode * required
`integer`
Model for Prompt generation (0 = openai, 1 = llmhub)

image_model * required
`integer`
Model for Image generation (0 = openai, 1 = nto, 2 = controlnet)

openai_api_key * required
`string`
OpenAI API key

llmhub_api_key * required
`string`
LLMHub API key

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** Schema `"string"`	*No links*
422	Validation Error Media type: application/json **Example Value** Schema `{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`	*No links*

GET `/status/{task_id}` Get Status

Parameters Try it out

Name	Description
task_id * required `string` *(path)*	task_id

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls `Accept` header. **Example Value** Schema `"string"`	*No links*

Code	Description	Links

422	Validation Error
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"detail": [
	{
	"loc": [
	"string",
	0
	],
	"msg": "string",
	"type": "string"
	}
	]
	}
		No links

GET `/result/{task_id}` Get Result

Parameters Try it out

Name	Description
task_id * required `string` *(path)*	task_id

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls `Accept` header. **Example Value** Schema "string"	*No links*

422	Validation Error
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"detail": [
	{
	"loc": [
	"string",
	0
	],
	"msg": "string",
	"type": "string"
	}
	]
	}
		No links

Schemas

Body_generate_image_generate__post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
llm_mode*	**Llm Mode**	[...]
image_model*	**Image Model**	[...]
openai_api_key*	**Openai Api Key**	[...]
llmhub_api_key*	**Llmhub Api Key**	[...]

}

Body_generate_image_with_llmhub_controlnet_generate_llmhub_controlnet_post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
llmhub_api_key*	**Llmhub Api Key**	[...]

}

Body_generate_image_with_openai_controlnet_generate_openai_controlnet_post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
openai_api_key*	**Openai Api Key**	[...]

}

Body_generate_image_with_openai_nto_generate_openai_nto_post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
openai_api_key*	**Openai Api Key**	[...]

}

Body_generate_image_with_openai_openai_generate_openai_openai_post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
openai_api_key*	**Openai Api Key**	[...]

}

Body_generate_image_wwith_llmhub_openai_generate_llmhub_openai_post {

image*	**Image**	[...]
mealplan*	**Mealplan**	[...]
llmhub_api_key*	**Llmhub Api Key**	[...]
openai_api_key*	**Openai Api Key**	[...]

}

HTTPValidationError {

detail	**Detail**	[...]

}

ValidationError {

loc*	**Location**	[...]
msg*	**Message**	[...]
type*	**Error Type**	[...]

}

Glossary

Classifier-Free Guidance (CFG) A technique used in diffusion models to control the strength of conditioning inputs by interpolating between unconditional and conditional predictions. Higher CFG values enforce stronger adherence to the prompt.

ControlNet An extension to diffusion models that introduces additional conditioning inputs, such as edge maps or depth maps, to guide and constrain image generation with greater structural precision.

Elo Rating A relative ranking method originally developed for chess, adapted here to compare image generation outputs by assigning scores based on pairwise comparisons.

FastAPI A modern Python web framework used to build the backend of the system. It facilitates fast, asynchronous request handling for serving image generation APIs.

GPT-4o A multimodal large language model by OpenAI that supports text and image input/output. Used in this project to generate natural language prompts from structured meal plans and to produce image-based transformations.

InstructPix2Pix A diffusion-based model fine-tuned for image editing through natural language instructions. It enables realistic transformations of images by interpreting and applying changes specified in textual prompts.

Large Language Model (LLM) A deep learning model trained on vast text data to perform tasks such as language generation, summarization, translation, and prompt construction. Used in this thesis for semantic interpretation of meal plans.

Likert Scale A psychometric scale commonly used in surveys to capture subjective ratings. In this thesis, it was used to evaluate the quality and realism of generated images.

Null-text Inversion A diffusion-based image editing technique that modifies the null-text embeddings to allow for precise, localized edits in an image without changing the global structure.

Prompt Engineering The practice of crafting precise and effective natural language prompts to guide AI models toward desired outputs, particularly important for controlling semantic image transformations.

Bibliography

[1] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4401–4410.

[2] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022. arXiv: 2204.06125 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2204.06125.

[3] A. Bandura, "Health promotion by social cognitive means," *Health Education & Behavior*, vol. 31, no. 2, pp. 143–164, 2004.

[4] S. Michie, M. M. van Stralen, and R. West, "The behavior change wheel: A new method for characterising and designing behaviour change interventions," *Implementation Science*, vol. 6, no. 1, p. 42, 2011.

[5] T. Brooks, A. Holynski, and A. A. Efros, *Instructpix2pix: Learning to follow image editing instructions*, 2023. arXiv: 2211.09800 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2211.09800.

[6] L. Zhang, A. Rao, and M. Agrawala, *Adding conditional control to text-to-image diffusion models*, 2023. arXiv: 2302.05543 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2302.05543.

[7] R. Mokady, A. Hertz, K. Aberman, Y. Pritch, and D. Cohen-Or, *Null-text inversion for editing real images using guided diffusion models*, 2022. arXiv: 2211.09794 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2211.09794.

[8] OpenAI, *Introducing gpt-4o's image generation capabilities*, https://openai.com/index/introducing-4o-image-generation/, Accessed: 2025-06-03, 2024.

[9] Z. He, W. Zuo, M. Kan, S. Shan, and X. Chen, *Attgan: Facial attribute editing by only changing what you want*, 2018. arXiv: 1711.10678 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.10678.

[10] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, *Stargan v2: Diverse image synthesis for multiple domains*, 2020. arXiv: 1912.01865 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1912.01865.

[11] L. Li, J. Bao, H. Yang, D. Chen, and F. Wen, *Faceshifter: Towards high fidelity and occlusion aware face swapping*, 2020. arXiv: 1912.13457 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1912.13457.

[12] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman, *Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation*, 2023. arXiv: 2208.12242 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2208.12242.

[13] A. Yang, A. Li, B. Yang, *et al.*, *Qwen3 technical report*, 2025. arXiv: 2505.09388 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2505.09388.

[14] RunDiffusion, *Juggernaut xl v9*, https://huggingface.co/RunDiffusion/Juggernaut-XL-v9, Accessed: 2024-05-01, 2023.

[15] A. E. Elo, *The Rating of Chessplayers, Past and Present*. Arco Publishing, 1978.

[16] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

List of Listings

List of Tables

List of Figures