

LVGL UI Detector

This machine learning project is a user interface widget detector for the [Light and Versatile Graphics Library \(LVGL\)](#). The base is the [YOLOv8 model](#), trained on synthesized datasets created from the programs/scripts in this repository and corresponding submodules.

The purpose of the model is to detect and locate widgets from a provided screenshot to aid in automated testing procedures requiring this information.

It was trained specifically for the LVGL framework, with the following widgets implemented in the dataset generator:

- Arc
- Bar
- Button
- Buttonmatrix
- Calendar
- Checkbox
- Dropdown
- Label
- Roller
- Scale
- Slider
- Spinbox
- Switch
- Table
- Textarea
- ...(more to come in the future)

This repository serves as the main entry point for the project and contains the main documentation as well as the project structure.


It also features a docker image for development purposes to work inside Jupyter notebook.

It additionally contains a few scripts for the container and also for inspecting or uploading datasets manually.

The image and scripts are **for my personal use** and will be deprecated in the future.

Project structure

The project is divided into several submodules, each serving a specific purpose:

- [UI Generator \(Version 1\)](#): A modified version of the LVGL simulator for PC that generates a single random UI and captures a screenshot with annotations, using C and LVGL. This generator was originally used for UI screenshot generation, but was **deprecated** in favor of **version 2**, which uses micropython as a base. ( [Documentation](#))

- [UI Generator \(Version 2\)](#): An updated version of the UI generator that uses micropython and corresponding LVGL bindings as a base. This version is more flexible, easier to use and more maintainable than the original version.
- [UI Randomizer](#): A python script module which uses the UI generator to create multiple UI screenshots with annotations in a single CLI interface. The output is organized into a dataset in YOLO format for training the model. **This module is generally deprecated** in favor of the UI Detector submodule, which combines the functionality of the UI Randomizer with YOLO training and optimization pipelines into a single repository.
- [UI Detector](#): A repository containing multiple scripts/pipelines for generating datasets, training the YOLOv8 Model and hyperparameter optimization. **This repository is the main focus of the project.** It relies heavily on the usage of [ClearML](#) for experiment tracking, model optimization and dataset versioning.
- [Paper](#): The repository containing the [exposé](#) and [final paper](#) for my bachelor thesis, describing the theoretical background, the project structure and the results of the project. Results include the evaluation of the base model and the project as a whole. Since this project and its submodules are subject to change in the future, [a list of commits is provided in the paper repository](#), which showcase the state of the project at the time of finishing the paper. The final source state of the project from the perspective of the paper can also be found in the [release marked paper-final](#) of this repository.

Dependency management

All projects use [Poetry](#) for python package creation and dependency management. This repository as well as each submodule contains a `pyproject.toml` file with the necessary dependencies to work with the project. **Please refer to the README of each submodule for more information on how to set up the environment.**

The main project has a `pyproject.toml` due to historic reasons, since it used to contain the code of the UI Detector submodule. It now serves more as a placeholder and starting point if new submodules would be added. In general, **the main project's `pyproject.toml` should not be used for setting up an environment**, but rather serve as a documentation and starting point for the project. In the future (*after the paper is released*), contents of the main project will get more cleaned up.

ClearML

The project uses [ClearML](#) for experiment tracking, model optimization and dataset versioning. ClearML is a powerful tool that allows for easy tracking of experiments, datasets and models. It also provides a powerful hyperparameter optimization tool that can be used to optimize the model for the best possible performance.

Since the [UI Detector submodule](#) relies so heavily on ClearML, it is required to have an account and API key setup to use it. This is also the reason why the [UI Randomizer submodule](#) exists, since it can be used to run the generator repeatedly without the hard dependency on ClearML. For training purposes, the [YOLOv8 python package](#) as well as [CLI tools](#) can be used to train a model on generated datasets without requiring ClearML.

ClearML also provides the possibility of setting up a standalone server, which can be used to host locally. If you do not want to setup an account with ClearML directly, you are required to setup a local server to use the UI Detector submodule. You can find out more information on how to set up a local server in the [ClearML server documentation](#).

ClearML configuration

For simply using ClearML, the API keys can be provided directly in the CLI, in code or as environment variables. To connect to the ClearML SDK it is as simple as running `clearml-init` in your terminal and following the instructions. Checkout the [Getting Started Tutorial](#) for more details.

However, to effectively orchestrate tasks/pipelines you need to use the ClearML agent, which requires a configuration file.

An example ClearML configuration can be found in the [ClearML agent repository](#), which contains a basic `clearml.conf` file. This file can be modified and stored in the target machine, mainly to configure the [ClearML agent](#).

More information about the configuration file can be found in the [ClearML config documentation](#).

Configuration options that need to be modified on the template are:

- `api.api_server`: The URL of the ClearML server you are using. If you are using the ClearML cloud service, this should be `https://api.clear.ml`, otherwise it should point to the URL of your local server.
- `api.web_server`: The URL of the ClearML web server you are using. If you are using the ClearML cloud service, this should be `https://app.clear.ml`, otherwise it should point to the URL of your local server.
- `api.files_server`: The URL of the ClearML files server you are using. If you are using the ClearML cloud service, this should be `https://files.clear.ml`, otherwise it should point to the URL of your local server.
- `api.credentials`: An object containing the `access_key` and `secret_key` for your account on the used ClearML server. They can be found/created in your ClearML account settings on the web interface. You may also configure these by setting the `CLEARML_API_ACCESS_KEY` and `CLEARML_API_SECRET_KEY` environment variables. More information on setting up these keys can be found in the [ClearML WebApp documentation](#).

Beyond that, it is recommend to checkout the `agent` section of the configuration file to configure the agent according to your needs. The agent is used to run experiments on remote machines and can also be used to run the UI Detector submodule on a remote machine.

(For the generator it requires a setup display server, such as the [X window system](#))

LVGL UI Generator (Version 1)

Version 1 of the LVGL UI Generator is a modified version of the LVGL simulator for PC that generates a single random UI and captures a screenshot with annotations. It is based on the [lv_port_pc_vscode](#) repository, since that is the IDE I use and also the only one I dared to modify.

When launched with the proper arguments, it will open a window with a single container widget of the provided size placed inside. After rendering the UI, it will generate a screenshot of the container and save it to the provided path. It will also generate a text file with the bounding boxes of the placed widgets.

Further information can be found in the [UI Generator v1 README](#).

LVGL UI Generator (Version 2)

Version 2 of the LVGL UI Generator is an updated version of the UI generator that uses micropython and corresponding LVGL bindings as a base. This version is more flexible, easier to use and more maintainable than the original version.

It has two modes of operation:

- **Random mode:** Generates a random UI with a specified number of widgets placed on a white background. It requires a provided list of widget types to randomly choose from.
- **Design mode:** Generates a UI based on a provided JSON design file. The design file describes the whole window, including styles, widgets and certain properties. There is a special `random` widget, which can be used to randomize widget creation in certain areas of the design. This mode is useful in creating more realistic looking user interfaces, as the random mode does not accomodate for styles regarding the containers.

Further information about usage and setup can be found in the [UI Generator v2 README](#).

UI Randomizer

The UI Randomizer is a python script module which uses the UI generator to create multiple UI screenshots with annotations in a single CLI interface. The output is organized into a dataset in YOLO format for training the model. Additionally, the script can handle uploading of these manually created datasets to [ClearML](#), if the specific `c1earm1` options are provided in the CLI.

Further information can be found in the [UI Randomizer README](#).

UI Detector

The UI Detector is a repository containing multiple scripts for generating datasets, training the YOLOv8 Model and hyperparameter optimization. **This repository is the main focus of the project.** It relies heavily on the usage of ClearML for experiment tracking, model optimization and dataset versioning.

Further information can be found in the [UI Detector README](#).

Docker image

A docker image is provided for development purposes. It is derived from the official [Ultralytics docker image](#) and contains all necessary dev-dependencies for the project (*i.e. JupyterLab*).

The image can be built using the provided `Dockerfile` in the root directory of the repository. Additionally, there's a [run container.sh script](#) provided for convenience in the [scripts](#) folder.

The script and image currently only serve development purposes for my own personal use and are not taking any other needs into account.

VSCode workspace

The project was developed using [Visual Studio Code](#).

The main repository contains the `ui-detector.code-workspace`, which organizes all submodules into separate workspaces. All deprecated submodules are still included in the workspace, but are commented out to clean up the workspace view.

The workspace file is intended for development purposes and suggests the usage of a few extensions. These included suggestions are generally required to develop the submodules properly.

► Required extensions for the workspace

- [Python](#)
- [Jupyter](#)
- [C/C++ Extension Pack](#)

I personally use a few more extensions than that and to credit their creators/maintainers, a purely optional list is provided here as well.

► Optional extensions for this workspace (*in no particular order*)

- [Project Manager](#)
 - [Todo Tree](#)
 - [Zotero LaTeX](#)
 - [Citation Picker for Zotero](#)
 - [LaTeX Workshop](#)
 - [LaTeX Utilities](#)
 - [Icons for Visual Studio Code](#)
 - [GitHub Copilot](#)
 - [Docker](#)
 - [Git Graph](#)
 - [Git History](#)
 - [Local History](#)
 - [MemoryView](#)
-

Author's Note

This project was created for my bachelor thesis at the [University of Applied Sciences Technikum Vienna](#). The thesis is titled "Precision at Pixel Level: You only live once doing UI test automation" and is available here: [Thesis](#)

Rationale for the Title

The subtitle "*You Only Live Once Doing UI Test Automation*" is a thoughtful homage to the *You-only-look-once* model, which is utilized for its efficiency and accuracy in real-time object detection. In the context of this research, it underscores the critical importance of precision in UI test automation for nurse call systems. The phrase "You Only Live Once" conveys the idea that in high-stakes environments, such as healthcare, there is little room for error. This highlights the necessity for accurate and reliable testing methodologies to ensure system robustness and reliability, effectively communicating that getting it right the first time is paramount.

Milestones

- ✓ Create UI generator
- ✓ Create UI randomizer
- ✓ Create model
- ✓ Make model predict the pre-defined widget list
- ✓ Finish Exposé
- ✓ Finish Exposé presentation
- ✓ Complete README documentation in all submodules
- ✓ Finish bachelor thesis
- ✓ Finish thesis presentation
- ✓ Add API documentation
- ✓ Did not catch burn-out syndrome

Feature roadmap

- ✓ Add more widgets to the generator and model
- ✓ Improve dataset size (min. 50 per class) and training process (1-click-pipeline)
- ✓ Add widget customization in generator (color, size, etc.)
- ✓ Add design file system for randomizer & generator

License

This project is licensed under the MIT License - see the [LICENSE](#) file for details.