TensorFlow UI Coding Standards Document

Project by Try Catch Degree

Felipe Kosin Jorge	u17291195
Werner van Rensburg	u15118046
David Walker	u19055252
Wessel Kruger	u18014934
Siviwe Lechelele	u18221409

GitHub

Branching Structure

The Master and Development branches are only to be used to store complete, working versions of the codebase, which can be deployed at any time. The Development branch is to be tested thoroughly and approved by the entire team before being pushed into the Master branch.

Beyond that, each subsystem is to have its own branch – examples include Frontend and Code Generation. Under each of those, a feature branch should be created to implement and test features before being added to the subsystem as a whole.

Merging Structure

All merges to an upstream branch should be performed by means of a pull request. Any pull requests which move into a branch in which another member has implemented specific changes should be reviewed by the member who has made those changes. In the case of merge conflicts, discussions should take place between members whose changes merge, to reach a cohesive, fully-working solution.

GitHub Actions are applied on Master and Development branches to ensure only compliant code is submitted to those branches. Should the Actions integration tests fail, the pull request is to be closed and the errors are to be resolved before another pull request is made.

Project management

Issues should be opened regularly for any items which need to be completed but which cannot be completed in short time spaces as part of a specific feature implementation. Should any one person be working on a subsystem which focusses on those items, they may be assigned the specific issue. Where possible, issues should be closed by referencing them from a specific commit.

In addition, the to-do board should be used to keep track of present progress on whichever tasks are currently being worked on, to ensure that no two people duplicate work unnecessarily.

Package Structure

Any non-intertwined subsystems should exist in separate top-level folders, such as the Frontend and Python-Docker subsystems.

Hence, as the project exists right now, the three top-level folders should be:

- Frontend, for Angular frontend development
- Python-Docker, for runnable code backend development
- Documentation, for the storage and sharing of PDF copies of documentation

Thereunder, inside the Angular project, each subsystem and each feature should exist within their own folder and package, to keep imports logical and tidy, and avoid namespace pollution.

Angular

All visual components should adhere to Google's Material Design guidelines to the greatest extent possible. To this end, if an Angular-Material component exists which fulfils a specific needed purpose, it should be used over and above any alternative native Angular components.

Furthermore, whenever possible, if equivalent functionality can be achieved with the use of one library, no additional libraries should be used, to minimise the number of dependencies and modules required for the project to compile and run.

Naming Guidelines

Folders

Folders which are used for organisation should be given descriptive names which are capitalised, and in which spaces are hyphenated for simple command-line based navigation.

Documentation

Documents should be named descriptively, with capital names and hyphenated spaces.

Angular

When coding in Angular, camel case is to be used for all object and variable names. Objects should be capitalised, while functions and variables' names should always begin with a lower-case letter.

Python

PEP conventions should be followed in Python. Hence, classes will use Pascal case, and variables, functions, and modules will use snake case.

Formatting conventions

Line breaks

A blank line should exist between all functions, to promote legibility. Furthermore, wherever possible, a line should not exceed 80 characters in length, for legibility and tidiness.

Indentation

Tabs should be used for indentation. Each level of braces should be one indentation level deeper than any containing braces.

Documentation guidelines

When functions are above 5 lines long or perform any activity which goes beyond the simple modification, storage, or retrieval of a value, comments should be applied to them. Multiline comments should be performed in a manner which is compliant with third-party generation systems such as Doxygen, meaning that even when languages accept single-asterisks comments, two asterisks should be used to define a comment.

The following details should be provided for such functions:

- A list of parameters, and a brief explanation of what each is or does.
- A brief description of the purpose of the function.
- A brief description of the output and intended return type of the function.

Where possible, annotations should be used to specify these details in a way that enables the language's systems to interpret the comments for IDE explanation.

Furthermore, documentation should be exported as a PF file whenever completed for viewing – no raw .tex or .docx files should ever be seen by anybody external to the group.

Testing Guidelines

All services and components should have two unit test implemented for each function, one to test the positive case and one to test the negative case. Furthermore, all systems which are intended to work together (i.e., "use" each other as specified in the SRS document) should have a test dedicated to checking their intercompatibility.