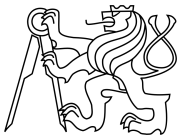
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Extension of a ML experiment management tool
Student:	Bc. Martin Chovanec
Supervisor:	Dipl.-Inf. Klaus Greff
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2017/18

Instructions

Improve Sacred, a tool for machine learning experiment management, to provide the users with API for collecting monitoring information and to present the information about both running and finished experiments on a dashboard.

1. Get familiar with the concept of machine learning algorithms.
2. Understand quality metrics of machine learning models and their importance for researchers.
3. Get familiar with Sacred, an open source tool that helps managing configuration and runs of experiments, and with the TensorFlow library for machine intelligence.
4. Implement a new Sacred API for collecting monitoring information from running experiments that would simplify using the API with TensorFlow. Demonstrate the API on an experiment that uses TensorFlow.
5. Create a dashboard where both running and finished experiments can be observed.
6. Include filtering and searching over the experiments by various parameters to the dashboard.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 19, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Master's thesis

Extension of a Machine Learning Experiment Management Tool

Bc. Martin Chovanec

Supervisor: Dipl.-Inf. Klaus Greff

8th May 2017

Acknowledgements

This project could not have been created without ideas and support from Klaus Greff from the Swiss AI Lab IDSIA, whose experiment management tool was being extended in this thesis. I would like to express my thanks to him for his classes on intelligent systems at the university in Lugano as well as for his supervision of the thesis. I thank also his colleagues from IDSIA and other institutes who participated in requirements elicitation and provided their feedback. In addition, thanks to Vladimír Kobetič for his software engineering comments on the text and other persons and institutions that the project directly or indirectly owes its existence to, but most importantly to my family for a quarter-century of support and patience.
Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042) is greatly appreciated.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that make use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Praha on 8th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology
© 2017 Martin Chovanec. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Chovanec, Martin. *Extension of a Machine Learning Experiment Management Tool*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Cílem této diplomové práce a souvisejícího projektu je rozšířit stávající nástroj Sacred, který pomáhá výzkumníkům v oblasti umělé inteligence spravovat konfigurace a zachovávat výsledky experimentů. Část projektu se zabývá přidáním nových funkcí do nástroje Sacred, které uživatelům umožní zaznamenávat metriky experimentů a zjednoduší práci Sacred ve spojení s výpočetní knihovnou TensorFlow. Sebrané informace uchovávané v databázi Sacred musí být přístupné uživatelům. Práce se proto rovněž zaměřuje na vytvoření webového monitorovacího panelu Sacred: Sacredboard.

Pro poznání potřeb uživatelů proběhly diskuse s hlavním vývojářem nástroje Sacred, jehož uživatelé byli následně osloveni formou dotazníku. Uvedené zdroje pomohly formalizovat požadavky pro Sacredboard i zamýšlená rozšíření pro Sacred. Zjištěným požadavkům byla přiřazena priorita, byly rozpracovány a implementovány. Díky veřejnému umístění projektu na portálu GitHub poskytli uživatelé první připomínky již krátce po vydání první verze. V dalších kapitolách je popsána softwarová architektura, postupy a nástroje použité v průběhu vývoje. Na závěr je představen ukázkový projekt využívající nové schopnosti programu.

Zatímco Sacred byl rozšířen o rozhraní pro záznam měření průběhu experimentů, Sacredboard byl vytvořen pro zjednodušení procházení záznamů z databáze Sacred a sledování průběhu experimentů. Přestože některé funkce zatím nebyly dokončeny a další by bylo vhodné v budoucnu doplnit, projekt byl již řadou uživatelů přijat. Někteří z nich začali program rozšiřovat o vlastní funkce. Vyjadřuji naději, že pomohou projekt udržet aktivní i nadále. Sacred a jeho rozšíření včetně Sacredboard byly přijaty jako příspěvek pod názvem “Sacred: How I Learned to Stop Worrying and Love the Research“ na prezentaci v tématickém celku o opakovatelnosti experimentů na konferenci o vědeckém výpočetním programování s jazykem Python, SciPy 2017, která se uskuteční v Austin, Texas.

Klíčová slova Sacred, Sacredboard, strojové učení, experiment, IDSIA

Abstract

The aim of this thesis and the accompanying project is to extend Sacred, an existing software tool that helps machine learning researchers with managing configuration and maintaining results of their experiments. A part of the project deals with adding new functionality to Sacred to provide its users with a standardized way of tracking experiment metrics and to simplify working with Sacred and the TensorFlow computational library. Collected information kept in the Sacred database must become accessible to the users. Thus, the thesis also focuses on creating a web dashboard for Sacred: the Sacredboard.

To learn about needs of researchers, discussion with the main developer of Sacred took place and a questionnaire was distributed among users of the existing tool. These sources of information helped to formalize requirements both for Sacredboard and the Sacred extension. The elicited requirements were prioritized, elaborated and the development began. Being publicly available on GitHub, users provided their first feedback soon after the initial release. In the next chapters the software architecture, techniques and tools that were used to support the development process are described. At the end, a sample project that uses the new features is presented.

To conclude, while Sacred has been extended with an interface to log measurements taken during experiments, Sacredboard has been created to simplify browsing the Sacred database and observing running experiments. Although some features have not been finished yet and others are desirable to add in the futre, the project has been already adopted by a number of users. Some of them have started with development of their own features. Hopefully, they will contribute to keep the project active. An abstract called "Sacred: How I Learned to Stop Worrying and Love the Research" on Sacred and its extensions, including Sacredboard, has been accepted for presentation as a talk in the Reproducibility track of the SciPy 2017 Scientific Computing with Python conference in Austin, Texas.

Keywords Sacred, Sacredboard, machine learning, experiment, IDSIA

Contents

List of Figures

List of Tables

Introduction

Machine learning is a subfield of computer science that deals with the study of learning algorithms. The main emphasis is put on automatic methods allowing to complete a task, to make accurate predictions or to behave intelligently without human intervention or assistance. Instead of explicitly programming the computer to perform the task, the program has to find a general way of solving the problem based on its previous observations and examples. [1] In simple words, it means that the computer is taught *what* should be done instead of *how* it should be done. For instance, when predicting stock market prices for the next day, the computer basically predicts the result by looking at the development of current stock prices and compares it with similar situations from the past. But there is nobody to write specific rules describing the possible situations. The computer recognises it implicitly.

However, making computers learn and perform non-trivial tasks without being explicitly programmed is a complex and time-consuming activity with uncertain results. In general, it involves choosing or creating a model capable of solving the given task, configuring the model's learning hyperparameters, gathering relevant training data and running a training process while evaluating its outcomes on a test set of data.

Conducting machine learning experiments is a dynamic, empirical process that brings many challenges of engineering and record tracking. In addition to seeing the actual results of an experiment, the machine learning researchers are also interested in the impact of the model chosen, its learning hyperparameters and the selection of training data on the performance and outcomes of the experiment. To compare and outperform its previous results, the experiment may be repeated several times while searching for a better learning algorithm. As the result of the experiment often depends on random values generated during the run, the training process is not always deterministic. It is extremely important to keep track of the many involved factors like random seeds for pseudorandom generators, hyperparameters, package dependencies, the evolution of the experiment source code, and the training data. Having

reproducible experiments, the algorithms can be systematically improved.

To configure, run and keep track of experiments, their configuration, results and even the random seed used to generate the pseudorandom values, the researches at the *Dalle Molle Institute for Artificial Intelligence* (IDSIA) [2] in Switzerland have started developing an open-source tool that simplifies the tasks. Since its initial release in 2014, *Sacred* [3] has spread out and became used by people outside IDSIA. As of beginning of 2017, it has been starred more than four hundred times on GitHub.

Sacred makes managing experiments for the user very easy. It is written in Python and for Python, which is probably the most used programming language in the machine learning field.[1] The tool adds a minimal configuration overhead and was designed to make researchers using it even under pressure of a deadline. By starting a Sacred-powered script, basic information about the host computer, the Python environment, and user-defined configuration is automatically collected, allowing to reproduce the experiment later. Another important feature allows storing the collected data to a database backend by attaching a Sacred "observer". The user of the library is free to add any relevant information to the database while the experiment is running.

Sacred, however, solves only a half of the problem and leaves inspecting the data an open question. The users must query the database directly as no convenient graphical user interface has been developed yet. The current situation is neither productive nor user friendly.

This thesis aims to identify the requirements and to implement a graphical user interface to browse the data collected by Sacred. Understanding what the researches need to learn from the experiments they run is essential in order to provide them with a useful view on the results. Sacred itself will have to be extended to achieve the goal.

Goals

The main objective of this thesis is to help researchers with monitoring and maintaining their experiments by providing them with a dashboard for Sacred – *Sacredboard*. Like Sacred, it will be provided as a standalone open-source application. Because the software itself should be maintainable and extensible by other developers after finishing the thesis, its structure should be easy to understand and well documented. The thesis itself should serve as a reading for future developers to become familiar with the purpose and architecture of the project.

[1] When searching for "machine learning" projects on GitHub, there were 10 679 Python projects + 4,766 Jupyter projects, compared to second Matlab with "only" 5 887 projects.

At first, getting at least a basic level of understanding of the concepts that the researchers are dealing with is advantageous. A number of books has been written on this topic, for example the classical *Pattern Recognition and Machine Learning* by Christopher M. Bishop [4]. Lectures given at universities, such as the Intelligence systems [5] course taught by Jürgen Schmidhuber [6] and his colleagues from IDSIA, are an advisable way of learning and practising the basics or even a little bit advanced topics in a clear and understandable manner. For those who remain untouched with the problematic, the field is shortly introduced in Chapter 1.

To become more familiar with the needs that researchers and current users of Sacred may have, training an own neural network model using the popular TensorFlow [7] framework and Sacred will be described in Appendix A. TensorFlow will be used to build the computation graph and execute the experiment, while Sacred will help to parametrize the program and observe its results.

The main developer of Sacred, Klaus Greff from IDSIA, will assist with elicitation of requirements for Sacredboard (see Chapter 3). He and some of his colleagues from and outside IDSIA use Sacred to do their research on a daily basis. To take advantage of these connections, a survey will be conducted among the users to learn more about their needs and expectations of Sacredboard. The results of the discussions and survey will be captured as user stories in Chapter 4, from which the functional and non-functional requirements for the application will be derived. The software architecture and design fulfilling the needs will be presented in Chapter 5.

The project will be hosted on GitHub, which is a popular open-source software hosting site that has been already used by Sacred. It allows interacting with users through a system of tickets (issues) and contributing to the source code base. Sacredboard versions will be released continually to the public as the project will proceed. An essential part of Sacredboard should be a usable user and developer documentation, because the project is likely to be extended by other developers after finishing this thesis. Ideally, a wiki or similar platform would be used to create both the user and development documentation.

Motivation

My personal motivation to work on this project has emerged from my earlier semester of study at *Università della Svizzera italiana* (University of the Italian Switzerland) [8] in the Swiss canton of Ticino.

One of the courses I took there was called Intelligent Systems [5], taught by a recognised expert Jürgen Schmidhuber [6] from IDSIA. His lectures were alternated with Jan Koutník and complemented by Klaus Greff's "TA sessions", where all uncertainties were explained to the students. The topic was

both interesting and well presented and it is regrettable that no such course was available at the Faculty of Information Technology at the Czech Technical University in Prague.

After coming back to Praha in early 2016, I started improving my German, a language that (like many others) distinguishes three grammatical genders of nouns. I was wondering whether the written form of German nouns can be used to determine their gender, and for that, I started conducting my own experiments using a neural network library developed at IDSIA. Because I asked for advices, we were in touch with some of the people from time to time. Later, we agreed on extending Sacred to help the researchers with their daily routine.

Chapter 1

Problem Domain

This chapter aims to briefly describe the terms used throughout this document and the typical workflow that researchers follow when conducting their experiments. The text is structured such that the readers that are already familiar with the field may skip to the next chapter without missing any important part. Those who need a gentle introduction to artificial intelligence and machine learning are advised to read the following sections.

1.1 Artificial Intelligence and Machine Learning

Artificial intelligence (AI) and machine learning (ML) are two terms often appearing together as if they were synonyms. In fact, most of the recent great improvements in artificial intelligence was accomplished thanks to machine learning. It is not only beating the best Poker players in the world, driving autonomous cars and natural language processing what is considered as a big success in the area. Machine learning can also help saving human lives, such as in medical detection of diseases [9].

But it is often argued that machine learning is in truth only one of the several approaches in achieving artificial intelligence. The following lines with a few examples should help with understanding the concepts and differences.

Artificial Intelligence

Artificial intelligence (AI), as a field in science and engineering, represents a wide variety of subareas that are connected with performing intellectual tasks. The Turing Test, proposed by Alan Turing in 1950, considered a machine to posses intelligence if a human interrogator was, in written communication, unable to determine whether his counterpart was a machine or a human. This kind of definition measures the success of achieving intelligent behaviour in terms of fidelity to human performance and closely couples intelligence

with the human mind, its way of thinking, problem-solving, learning[2] and acting.[3] Nevertheless, the human performance may not always be the ideal. Therefore, another, rational, approach defines AI as the study of computations that make it possible to perceive, reason and act[4] or as the study of the design of intelligent agents.[5] [10] On a more specific level, AI consists of reasoning and problem solving, knowledge representation, planing, learning, motion and manipulation (robotics), and others. [11] A program that can play chess or plan the (almost) optimal route for a travelling salesman can be considered to be intelligent, even though it may use a known algorithm and was explicitly programmed to perform such task.

Machine Learning

Machine learning (ML) is regarded as one of the subfields and enablers of artificial intelligence that allows computers to perform tasks that they are not explicitly programmed to. To draw a better distinction line between AI and ML, consider the problem of recognizing hand-written digits, represented as 2D images. This is an example of a *classification task*.

Classification Expect to have a large dataset of images, each containing one hand-written digit in the range of 0 to 9. None of the images looks exactly the same as any other. Even two images of the same digit written by a single individual would not look the same. Furthermore, the dataset contains images from different authors, making the variability of possible input representations even higher. Assigning the images the number they represent is a non-trivial intellectual task that could be solved by specifying hand-crafted rules and involving heuristics to scan the strokes of the written digits. This manual approach would, however, lead to a tremendous amount of rules, exceptions from the rules, exceptions from the exceptions, and would in general give poor results. [4]

Adaptive model Better results can be achieved by utilizing the machine learning techniques, such as by engaging an underlying adaptive model. This can be for instance a neural network or a model based on Bayesian statistics. Before the model is able to recognize the digits, its parameters must be adapted – *trained* – for the task. The principle behind the scenes is called *pattern recognition*. By discovering and exploiting regularities forming patterns in the training data, the essential information (in terms of the information theory) gets imprinted on the model. Thanks to the imprint, the model can recognise similar patterns in new data that it has possibly never seen before. In

[2] Bellman, 1978
[3] Kurzweil, 1990
[4] Winston, 1992
[5] Pool et al., 1998

practical applications, training data set is chosen such that its variability is large enough to embrace all the necessary patterns while keeping it as small as possible.

On a high level, adaptive models can be thought of as black-boxes that take inputs and produce outputs with a certain level of *accuracy*. Unlike conventional programs, they have a training mode in which their internal parameters get modified in order to increase the accuracy by minimizing the error on its outputs, or to improve various alternative criteria. These metrics have several names in the literature, but they are often interchangeable: the *loss function*, *error function*, *cost function* or the *objective function* (as something that should be minimized or maximized).

Supervised and Unsupervised Learning Learning can be either *supervised* or *unsupervised*.[6] In supervised learning, the model learns on data that has been already classified by someone else and it tries to discover common patterns that map the input data to the desired output. On the other hand, when not mainly interested in the actual value of the digits (for instance, given a large set of a relatively smaller number of different characters written by an unknown ancient civilisation thousands of years ago), at least trying to group them automatically based on their shape could help with archaeological research. Such task is an example of *unsupervised learning* because it is not explicitly declared how many different categories there are and the category for each of the characters is not known even by the researchers.

Regression Machine learning is not only about classification. Think about a system that predicts temperature for the next day based on weather changes during last 24 hours. This type of task is called *regression* [4] as the output is a continuous variable rather than a discrete category. The algorithms used for both types of the tasks are similar, yet the outputs must be treated differently.

Imagine having two models that take photographs as inputs. The first model is used to guess the temperature on the place where the photograph was taken. The second model can distinguish a cow from an aeroplane. We may decide to encode "a cow" as an 8 and "an aeroplane" as a 9 and we show a photograph of a cow gazing on a green meadow to the two models. Let us assume that the real air temperature on the meadow was 8,0 °C and the first model guesses 9,0 °C. It makes no big difference, the guess was close. But what if the classification model outputted 9 instead of 8? Either there is a cow or there is a non-cow; the difference between a plane and a cow is relatively bigger than between 8,0 °C and 9,0 °C. Obviously, when evaluating the accuracy of the two models, different methods must be used, such as the mean square error of the temperature differences for the first model and the percentage of correctly classified images for the second one.

[6] In fact, there are more types, e.g. reinforcement learning.

Deep Learning in Neural Networks

One of the currently most popular adaptive models is the neural network. It is an interconnected structure of simple individual processors, called *neurons*. In principle, a neuron takes an input vector of real values, multiplies it with a vector of internal *weights* and passes the computed value to an *activation function*, whose outcome is the output of the neuron. Such simple "network" is called *perceptron* and can be used for linear classification (e.g. to separate points in a space by a line or plane). The problematic part is to specify the plane, as no exact formula is given to the neuron. Instead, the points in the training set are shown to the network together with their correct classification (e.g. -1 and 1). Based on the difference between the actual and desired output values, the neuron weights are modified using the *backpropagation algorithm* until the network yields correct results. To make sure the model learned to generalize, another set of data, called the *test set*, is used to evaluate the performance.

Deep Neural Networks More difficult tasks, such as speech recognition or driving a car, cannot be solved by a single perceptron. Instead, a higher number of neurons is stacked across multiple layers. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons. When the network has more than just a few layers, we speak of a *deep neural network* (DNN). The deepest networks are, in a sense, *recurrent neural networks* (RNN) that connect the neurons not only in the forward direction from the input towards the output layer, but also recurrently to the same or a previous layer. This concept has been winning many international competitions since around 2000 in performing tasks like handwriting recognition and around 2010, deep networks even achieved superhuman performance. *Deep learning* is about accurately assigning the weights of such networks to make it exhibit the desired behaviour. [12]

Future of AI

So far, models need a large number of training data to make decisions, much more than a human. Neural networks may tell whether a patient has a certain disease or not, but they can rarely reason the decision, which might be as important as the decision itself. AI research in this area has the potential to be the most impactful technology that humankind has ever developed. [13]

1.2 Conducting Experiments

In the previous lines, the basic ideas that make computers behave smart have been covered. Nevertheless, the key to success in creating an intelligent system

consists in research across many dimensions, including collecting and preprocessing data, choosing or creating an architecture of the underlying model, implementing it and evaluating it.

1.2.1 Data Preparation

Datasets A lot of data is necessary to teach computers non-trivial tasks. Fortunately, there is already a large number of standard datasets available, ranging from visual data, speech and music samples to medical and biological data. [14] Using standard datasets contributes to doing high-quality research, since the computer scientists can exchange their experience more easily and compare their performance relatively to the same data base. Of course, not every problem has a standard dataset and in order for the program to perform its task, the data must be collected as a part of the study.

Having gathered and labelled the data, a subset is selected to perform the training on. As mentioned earlier, the *training set* should be relatively small while covering the most general cases. After finishing the training phase, the *test set* is used to evaluate the model's generalisation capability.

Feature Extraction Processing data in arbitrary form is still a computationally expensive task. For most practical applications, the original inputs are typically preprocessed into another format that makes the pattern recognition problem easier to solve. This pre-processing stage may involve resampling images or audio to a unified resolution, deskewing hand-written text, denoising, Fourier transform and so on. As far as we know, some models require a higher level of preprocessing than others. Remember that all the data, not only the training set, must be transformed using the same steps for the learning to work. [4]

1.2.2 Training a Model

Choosing and modifying the most suitable machine learning model can be a project on its own. There are several software libraries for different programming languages, most noticeably for Python and MATLAB, providing configurable implementations of well-known models. It might suffice to pick one of them, select a model, such as *decision tree*, *support vector machine* or one of the neural networks, and call a `train` method on the training set, followed by calling a `run` method on the test set for evaluation of its generalisation capability. This approach would only result in a decent performance in case of being lucky enough to just a relatively simple task being solved.

Normally, once the type of the adaptive model has been chosen, it must be adjusted to fit the problem needs. This involves configuring learning *hyperparameters* that decide the algorithms and their settings to be used for the training. Namely neural networks require more than that. As they are made

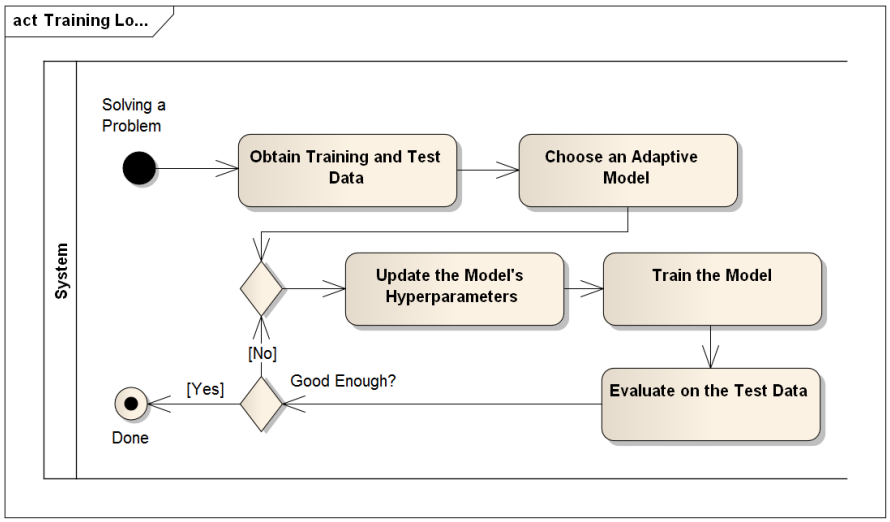


Figure 1.1: Training Loop

of individual neurons, the units must be organised to a network in a structured and scalable way based on the type of data they should process. There is a research just in exploring the network architectures and their benefits for particular research areas. Even after opting for a network architecture, specifying the training algorithms and their settings, the network structure must be adapted, for instance the numbers of neurons in each layer, the number of the layers, number of inputs, number of outputs, etc. To make the circumstances even more complicated, the selected model can be comprised of several other models.

Unfortunately, hyperparameters and configuration possibilities may be many and finding a valuation that yields superior results, ideally in a reasonable time, is difficult. Although there are several rules of thumb helping with the decision, the search space remains large. Thus, training (and, consequently, model construction) is an iterative process of patient hyperparameters modification, training, testing, evaluation, and repeating the whole process as long as improvements are foreseeable. It is sometimes referred to as the Training Loop, as depicted in Figure 1.1.

Sacred has been designed to keep track of iterations, or, more precisely, of the individual training "rounds", referred to as *runs*. For each experiment, researchers may compare the runs and their performance in terms of model precision, accuracy, loss, or speed. As with any other optimization task, the possibly high amount of computing resources is one of the reasons why we do not want to lose the results. In fact, large neural networks are nowadays being trained on GPU clusters and both training and using the network to generate some results can take significant time, from minutes to weeks or even more.

Chapter 2

Sacred

Sacred has been created at IDSIA to configure, organise, log and reproduce experiments written in Python. As mentioned in the Introduction, Python is among the most used languages for machine learning. It can be attributed to the relatively high number of libraries available for computations and to the syntax of Python is much more data-oriented than the syntax of C and co., which makes prototyping much easier. (In fact, most of the computational libraries do internally use effective C code, but the developers are abstracted from that.) Sacred adds another piece to the puzzle of machine learning ecosystem.

This section describes the "as-is" functionality of Sacred to support training process and the Sacred API and the information being collected. For consistency, the term *experiment configuration* (or simply *configuration*) refers to every configurable parameter of the training process, including hyperparameters, learning algorithm settings and similar. [3]

2.1 Supporting the Process

The key features provided by the tool are:

- keeping track of all the parameters of experiments
- running experiments with different settings
- saving configurations for individual runs in a database
- reproducing experiments

Sacred achieves this through the following main mechanisms:

- **Config Scopes:** A convenient way to define experiment configuration. When multiple scopes are defined, users may choose the one (or a combination of those) they want to use.
- **Config Injection:** Parameters defined in ConfigScopes are implicitly available in every function of the experiment.

- **Command-line interface:** Sacred extends possibilities of running experiments by enriching them with a command-line interface that makes the choice of setting individual parameters easier.
- **Observers:** Observers periodically check the experiment status and may log all kinds of information, including experiment dependencies, configuration used and results to a database. The most important is the *Mongo Observer*, as it provides a flexible way of storing almost arbitrary structured data, it is the most widely used, and new features use to be available for it earlier than for the others.[15].
- **Automatic seeding:** Helps controlling the randomness in experiments, making the results reproducible

2.1.1 Running Experiments

Sacred experiments are programs written in the Python programming language. It does not imply other technologies cannot be used with Sacred at all, but we assume that a Python main file in which all the necessary configuration can be performed is available.

In the following part, Sacred's ability to manage experiments on a simple classification task is presented using the famous Iris flower dataset [16], which often appears in educational examples. It concerns three species of Iris (Iris setosa, Iris virginica and Iris versicolor) and the means of distinguishing among them. The classification is based on four features: the height and width both of the sepal petal leafs[7].

The code snippet in Listing 2.1 uses *Support Vector Machine* (SVM) as a model to determine the Iris specie. The first 90 records of the 150-row dataset serve to train the model and the remaining 60 to evaluate the mean accuracy of the predictions. The resulting score is written to the standard output, where it has to be collected by another program or a human to keep the records.

When running the experiment multiple times, it can be seen that the results of individual runs vary. This occurs due to the permutation being performed before training the model. It is, however, quite necessary: the first 90 records being used for the training are not guaranteed to equally represent all the three species. By randomly selecting individual measurements, we hope the training set will consist of representative samples.

We might want to find out whether a better accuracy can be achieved and start experimenting with different hyperparameter values while keeping the selection of the training data the same. Sacred can be incorporated by a none too complicated change of the code as in Listing 2.2. Such script is run in the same way as the original one: using `python file_name.py`.

[7] *sepal and petal leafs* = modified leafs forming the blossom part of flowers

Listing 2.1: Experiment without Sacred

```
# Set hyperparameters
C = 1.0
gamma = 0.7
# Load the dataset and permutate it
iris = permutate_randomly(datasets.load_iris())
# Set up SVM
clf = svm.SVC(C, 'rbf', gamma=gamma)
# Train on the first 90 records
clf.fit(iris.data[:90], iris.target[:90])
# Evaluate accuracy on the rest of the dataset
print(clf.score(iris.data[90:], iris.target[90:]))
```

Output: `0.95`
Based on an example on the Sacred project page [3].

Code Explanation The Sacred-enabled example in Listing 2.2 introduces a new variable `ex` representing the experiment and two methods decorated by Sacred annotations. The `ex.automain` annotation denotes the entry point for the experiment. Its parameters `C` and `gamma` are automatically set according to the `cfg` method decorated by the `ex.config` annotation. The main method returns a value that should be understood as the main result of the experiment. The advantages brought by this separation, and storing more detailed information, are described later in the text.

2.1.2 Observers

One of the key goals of Sacred is keeping track of and monitoring the experiments being run. For that reason, the *observer* monitoring interface has been provided.

Information accessible to observers include experiment metadata (start time, operating system, package dependencies), the source code of the experiment file(s), the actual configuration parameters, the standard and error outputs, and the experiment status, among others. Users have also a dictionary at their disposal, called `info`, which can be used for storing any serializable data of their choice.

There are already several implementations of observers: most noticeably the Mongo observer, periodically storing the progress to a MongoDB database. As MongoDB belongs to NoSQL databases, the database schema is not as strict as in the case of relational databases. In fact, collections in MongoDB can store almost any serializable data objects consisting of key-value pairs and arrays. Thus, records in one collection may have completely different

Listing 2.2: Experiment with Sacred

```
from sacred import Experiment
# Define the experiment name
ex = Experiment('iris_rbf_svm')

@ex.config
def cfg():
   C = 1.0
   gamma = 0.7

@ex.automain
def run(C, gamma):
  # Load the dataset and permutate it
  iris = permutate_randomly(datasets.load_iris())
  # Set up SVM
  clf = svm.SVC(C, 'rbf', gamma=gamma)
  # Train on the first 90 records
  clf.fit(iris.data[:90], iris.target[:90])
  # Evaluate accuracy on the rest of the dataset
  return clf.score(iris.data[90:], iris.target[90:])
```

Output:

```
WARNING - iris_rbf_svm - No observers have been added to this run
INFO - iris_rbf_svm - Running command 'run'
INFO - iris_rbf_svm - Started
INFO - iris_rbf_svm - Result: 0.9833333333333333
INFO - iris_rbf_svm - Completed after 0:00:00
```

structure, which almost perfectly fits the needs of storing various information for different experiments.

Nevertheless, some users prefer using SQL databases, and for them, the SQL observer has been created, but it can currently store only general information about the experiment in a structured way. The experiment-specific logging data is stored as serialized objects. The list of observers contains additional modules for other databases, a module for File Storage (which does not require any database backend) and observers notifying about the progress via instant messaging services. This might be useful to inform users when a long-running task has finished.

Using a particular observer can be enabled either programmatically or by a command-line option. A deeper explanation and the list of available observers can found on the documentation page [15].

2.1.3 Configuration

Sacred offers several means of managing the experiment configuration, including in-code statements, command line options and configuration files. Configuration parameters behave like user-definable variables – choosing the number of parameters, their names and structure is left up to the user. The most powerful mean of setting their values is using *Config Scopes*, which are regular Python functions decorated with `@ex.config`. All the variables declared in its scope and their values are collected and passed to the main function. In the example Listing 2.2, we have only seen simple key-value pairs, but any serializable data type, such as arrays and dictionaries, may be used. Being a regular function, standard Python statements can be used.

Thanks to the extraction of the configuration parameters, it is now much easier to run the script with different parameters. Starting the experiment with a different `gamma` hyperparameter is now as simple as running the following command: `python file_name.py with gamma=0.9`. In this case, the the default value of `C` will be used together with the updated value for `gamma`. Other possibilities of working with the parameters are described in the Sacred documentation [15].

2.1.4 Controlling Randomness

When conducting experiments, researchers often rely on pseudorandom number generators to yield sequences of numbers that need to come from a certain random distribution, but the individual values do not matter. It is useful for random draws from data sets, initialization of random vectors, such as neural network weights, or for making random decisions during training to escape non-prospective states of the search space. Hence, randomness is a desired feature of experiments.

In certain cases, generating a completely unknown random sequence is undesired and should be prevented. For instance, to analyse the influence of modifying different hyperparameters on the model's ability to learn, we want to make the initial pseudorandom shuffle of the dataset in Listing 2.2 always yield the same order of records. The sequence produced by a pseudorandom generator normally depends on the initial value, called *seed*, which can be the current time or another physical variable. As most generators allow setting a custom seed, it is possible to repeat the sequences by resetting the seed to a known value.

Sacred regards seed as a special configuration hyperparameter that can be passed to the experiment either directly in the Config Scope or using the command line: `python file_name.py with seed=123 gamma=0.001` This approach should work automatically for code that uses only Python's built-in `random` or `numpy.random` modules. Seeds of other libraries might have to be initialised programmatically first.

2.2 Sacred Architecture in a Nutshell

Sacred is a Python library that takes control over experiment execution. Once started, the program uses two parallel threads: the first thread is used to run the actual code, the second thread is used by observers. Before the experiment starts, the attached observers are notified. Database observers (e.g. the Mongo observer) create a new entry in the database for the current run, including its configuration, status and start time. Every 10 seconds or any other specified period of time, the attached experiment is checked by the observers, its standard output, current time (referred to as the heartbeat time) and the run's `info` dictionary is updated in the database. The `info` dictionary is also exposed to the experiment code, enabling it to add various additional information to the database. Experiment can be either in one of the following states: `RUNNING, COMPLETED, INTERRUPTED, FAILED, QUEUED, TIMED_OUT`. A detailed explanation of the data model, information collected and experiment states can be found in the project documentation [15].

Chapter 3

Sacredboard

Doing a research requires not only to perform experiments and store their results, but also to be able to make use of the data for deciding the future course of actions. So far, the Sacred users had to rely on standard database tools and custom queries and scripts when accessing the records. One of the important goals of this project is to eliminate such rather uncomfortable way of retrieving data, and to improve the user experience by developing a graphical interface to the tool: the Sacredboard.

Before starting with the development of Sacredboard, the answers to the following questions had to be found:

1. Who are the users of Sacredboard (and Sacred)?
2. What features do they need?
3. How should Sacredboard integrate with Sacred?
4. How should Sacredboard be deployed and accessed by its users?

3.1 Requirements Elicitation

Answering the first question seems to be simple. Sacredboard will share most of its user base with Sacred. The desired functionality, originally based on discussions with the main developer of Sacred, was supported by a questionnaire distributed among its current users, coming mainly from the IDSIA laboratory, other research groups, and users who visited the Sacred project page on GitHub. These are mostly highly educated people that know a lot both about their needs for their research and about the software environment they use. Therefore, they should be able to help not only with identifying the functional requirements, but also with deciding the architecture and other non-functional requirements. Another mean of requirement elicitation was a self-exploration based on conducting an own experiment using Sacred and the TensorFlow [7] framework from Google (Appendix A).

3.1.1 Discussion

In discussions with the main developer of Sacred was concluded that it would be beneficial for Sacredboard to be built on the same technology as Sacred, which was written in Python, as it would require a little to no extra effort for the users to install the new tool in their existing environment. Sacredboard was, threrefore, proposed as a standalone Python application that would run on a host computer and would be accessed via a web browser. This approach, which is familiar to Python programmers from other applications,[8] has been found valuable because it enables multiple modes of deployment: the users may choose to run the application locally and access it directly from an automatically opened web-browser window, or to run its integrated HTTP server on a remote machine and connect to it from their workstation.

The integration between Sacred and Sacredboard, that is, the way of accessing the Sacred data from Sacredboard, must deal with the simplicity of the current framework, which is devoid of any application server that could act as an integration point. Instead, scripts that utilize Sacred interact directly with a database backend of the user's choice. The variety of backends that is Sacred compatible with, ranging from document-oriented to relational databases, demands consideration of their support in Sacredboard. Despite inconveniences that may arise when realizing the integration on a database level, it is, in this case, considered the most reasonable option. After all, Sacredboard should display data in the first place, and not directly interact with running experiments.

Sacredboard should provide its users with an overview of experiments and their runs. Next to a list of running and finished experiments identified by their names, the users should be able to follow the standard output of experiments, their hyperparameters and current results. Furthermore, the interface should simplify launching external monitoring tools, such as TensorBoard for TensorFlow [7], which may give even more detailed information about the progress of training and the model used, as well as charts to compare the results with previous runs of the experiment. Another necessary feature is to allow filtering of the list by different configuration (hyper)parameters in order for relevant records to become quickly accessible.

3.1.2 Questionnaire

The requirements premised on the discussion had to be confirmed with Sacredboard's potential users. A few researchers from the IDSIA and other laboratories were asked to complete a questionnaire whose aim was to learn about their habits with respect to using Sacred.

The most important goals were to determine which database backend should be supported by the new tool in the first place, whether the proposed

[8] Jupyter Notebook, TensorBoard

deployment solution would be suitable for the users, and, of course, to collect ideas for Sacredboard's features.

The questionnaire was created in Google Forms, sent by email to researchers that were in touch with the IDSIA, and it was also published on the Sacred project page. After a month being open, in total 16 people responded, of which 4 have declared they were not using Sacred at all.

Respondents were asked the following questions. The numbers in parentheses represent the number of votes for the particular answer:

1. How often do you use Sacred? *(single-choice question)*
   - not at all (4)
   - for some of my experiments (2)
   - for most of my experiments (6)
   - for all of my experiments (4)

All of the 6 respondents who used Sacred only rarely or not at all have further expressed their willingness to become a regular user if there was a graphical dashboard available. Those who did not work with Sacred were allowed to speculate how they would use it.

2. What kind of setup do you use? *(not at all / sometimes / often)*
   - I run Sacred locally (5 / 5 / 6)
   - I run Sacred on a remote machine (2 / 1 / 13)

This question was important because of the requested integration between Sacred and the TensorFlow framework. TensorFlow is capable of producing its own monitoring logs that are stored on the file system. Their visualisation is realized by the TensorBoard monitoring tool, which is requested to be launched directly from Sacredboard. For that, Sacredboard has to provide TensorBoard with the path where the logs are stored. If we had found that Sacred was run mostly on a local machine, we could have expected the logs to be stored mostly on the same computer. In contrast, if the files are stored on a machine other than Sacredboard is running on, thy are not guaranteed to be directly accessible from the new monitoring tool.

3. What do you use Sacred for? *(multiple-choice question combined with the possibility of a custom answer)*
   - Training neural networks (14); *(predefined option)*
   - Optimization tasks (1); *(custom answer)*
   - Downloading datasets (1); *(custom answer)*
   - Others (1); *(predefined option)*

The usage of Sacred was asked to ascertain that the questionnaire targeted appropriate users. Using it for downloading datasets was a surprising discovery, nevertheless, rather irrelevant.

4. Do you use Sacred observers (Mongo, SQL, ...)? *(single-choice question)*
    - Yes (12)
    - No (4)

Observers are an essential feature of Sacred, because they are responsible for storing experiment results. Two of the four respondents who did not automatically observe their experiments were those who did not use Sacred at all. These and the two others, who probably only configured their experiments with Sacred, were therefore not very valuable for the investigation.

5. What Sacred observers do you use and how often? *(not at all / sometimes / often)*
    - Mongo (2 / 3 / 7)
    - File Storage (6 / 5 / 1)
    - SQL Observer (10 / 2 / 0)
    - TinyDB (11 / 0 / 1)
    - Slack (9 / 2 / 1)
    - Other (11 / 0 / 1)

MongoDB is the recommended way of storing experiment results and the questionnaire has proved it to be clearly the most employed Sacred observer among the respondents. It is a logical consequence of the necessity of storing arbitrary structured data, which makes a document-oriented database a preferred choice. This result confirms that the support of backends in Sacredboard should prioritize MongoDB. However, the tool should be open for future extensions.

6. If using a database backend, do you typically use a shared database?
    - Yes (6)
    - No (6)

Users that connect to a shared database are more likely to request a certain level of isolation of their experiments from the others. This was a concern for one of the feature proposals in question 8.

7. Is the database accessible from your local machine?
    - Yes (10)
    - No (2)

Sacredboard needs a connection to a database backend for its operation. If the database machine was not directly accessible, working with the dashboard would require additional steps to overcome the limitation, such as running it on another machine. Sacredboard should support deployment on a server in the future, however, making it work locally seems to satisfy most of the users for now.

8. How important are / would be these factors for you when using the dashboard? *(multiple-choice question; scale 0 – 4, each point counted as a vote. 0 = I do not need it at all, 4 = It is a must have)*
    - Getting an overview of finished experiments (50)
    - Getting an overview of running experiments (48)
    - Accessing the details of the experiments (configuration, standard output, the "info" section) (44)
    - Displaying plots of error functions, accuracy, etc. (39)
    - Easy installation / setup on a local machine (33)
    - Adding notes or comments to experiments (28)
    - Deployment on a server where it could be accessed by multiple users (22)
    - TensorFlow: Starting Tensorboard for a single experiment run (21)
    - Starting experiments from the dashboard (18)
    - TensorFlow: Starting Tensorboard for a set of selected experiment runs (18)
    - Displaying the source code of the experiment (17)
    - Browsing experiments from different databases at once (15)
    - When using a shared database, protect "my experiments" from my colleagues (12)
    - A mobile user interface (assumed that sacredboard runs on a computer accessible from the phone) (10)

The above mentioned results confirm that users mostly expect the dashboard to allow them browsing their past experiments, their configuration and results, as well as monitoring experiments that are currently running. This proves that many of the experiments are long-running. The respondents have also supported with a high number of votes the proposal of displaying charts showing the progress of training. For that, it is necessary to propose a standardized model for capturing and storing sequential data in Sacred.

Among other features proposed by the respondents have also appeared a few interesting ideas:

- Comparison of configuration and results of multiple experiment runs to support hyperparameter optimisation
- Deriving new experiments from existing ones

The results were transformed into user stories and further elaborated.

Chapter 4

Requirement Analysis

This chapter deals with analysis of the information collected during requirements elicitation phase (in Section 3.1). The first part summarizes the gathered users' needs in the form of user stories, each addressing one of the needs and briefly describing the benefits for the user. Based on that, the scope of the thesis is defined and more detailed functional and non-functional requirements are elaborated. Each of the functional requirements can be traced back to its user stories. The end of the chapter is dedicated to analysis a few non-trivial scenarios in the form of use cases to specify the desired behaviour in a sequence of interactions between the user and the system.

4.1 User Stories

The following users stories reflect the needs discovered in the previous chapter. Although the actors of Sacred and Sacredboard could be theoretically divided into those who prepare the experiments, those who run the experiments, and those who evaluate the results, all the three roles are represented by a single actor, the *researcher*, in the document. This decision reflects the fact that it is always the person doing the research who benefits from the system.

US 1 **Overview of Experiment Runs (Running)**
A researcher wants to list experiment runs that are currently running so that he knows whether the result of the run is ready to be evaluated.
Priority: High

US 2 **Overview of Experiment Runs (Finished)**
A researcher wants to list experiment runs that have finished so that he knows what has been experimented on so far.
Priority: High

US 3 **Sorting Experiment Runs**
A researcher wants to sort the list of experiment runs by its attributes

so that he can find the results more easily.
Priority: High

US 4 **Filtering Experiment Runs (Numerical Values)**
A researcher wants to limit the list of experiment runs to runs that have a certain property equal, less (or equal) etc. than a given value so that he gets only results that are better (or worse) than the reference value.
Priority: High

US 5 **Filtering Experiment Runs (String Values)**
A researcher wants to limit the list of experiment runs to runs that have a certain property equal, not equal or containing another string so that he can filter the results e.g. by the experiment name.
Priority: High

US 6 **Filtering Experiment Runs (Date Values)**
A researcher wants to limit the list of experiment runs to runs that have a certain property on, before, after or on another date than the given date so that he can filter the results e.g. by the experiment start time.
Priority: Medium

US 7 **Filtering Experiment Runs (Experiment State)**
A researcher wants to limit the list of experiment runs to those that are in a particular state (see table 4.1 without the need of manually entering the query so that he can easily view e.g. only currently running experiments or those that have failed.
Priority: Medium

US 8 **Displaying Experiment Run Configuration**
A researcher wants to see the configuration of an experiment run so that he can relate the run results to the values that have been used to accomplish it.
Priority: High

US 9 **Displaying Experiment Run Extended Information**
A researcher wants to see the extended information stored in the "info" section of an experiment run so that he can access and learn the auxiliary user-defined information that has been measured or stored when the experiment was running.
Priority: High

US 10 **Displaying Experiment Run Output**
A researcher wants to see the standard output of the experiment run as it is running so that he has a better insight into the experiment progress.
Priority: High

US 11 **Launching TensorBoard for an Experiment Run**
A researcher wants to launch TensorBoard for a particular experiment run that was using the TensorFlow framework so that he can view the computation graph or advanced charts related to the run.
Priority: High

US 12 **Launching TensorBoard for Multiple Experiment Runs**

A researcher wants to launch TensorBoard for multiple experiment runs that were using the TensorFlow framework so that he can view the computation graphs and advanced charts of all the runs of his choice at once.
Priority: Medium

US 13 **Displaying Metrics Charts**
A researcher wants to monitor various metrics of experiment runs (such as accuracy over several iterations of a run) even without the need rely on TensorFlow or TensorBoard so that he can visually inspect the progress of learning.
Priority: Medium

US 14 **Commenting Experiment Runs**
A researcher wants to add notes to experiment runs (and read them) so that he can later recall what and why he has done in the run more easily.
Priority: Medium

US 15 **Displaying Source Code**
A researcher wants to see the source code in the state it was when the experiment run started so that he can use the code again or compare it with another version.
Priority: Medium

US 16 **Deleting Experiment Run**
A researcher wants to delete experiment runs that he does not need any more so that he does maintain only information relevant to him.
Priority: Medium

4.2 Scope of the Thesis

The user stories described in Section 4.1 represent the most valuable features that users need from Sacredboard. The scope of the thesis is to analyse all of them and implement all the high-priority ones together with a few medium-priority ones.

4.3 System Requirements

The aim of requirements analysis is to discover the bounds of the software and to detect any overlaps or conflicts. They are divided into functional requirements addressing the user needs directly and non-functional requirements that support them.

4.3.1 Functional Requirements

The requirements listed below are made of "shall" statements that describe the requested functionality and assumptions known or elicited during the requirement analysis phase.

Requirements for Sacredboard

F 1 **List of Experiment Runs**
Realizes: User Stories US 1, US 2
Sacredboard shall display a table of experiment runs in the database. The "run table" shall include columns for the experiment name, the state of the experiment run, the start time of the run, the last activity time (heartbeat), the machine name on which the experiment is or was running (hostname), and the *run result*. States shall be represented as coloured icons according to the specification in table 4.1.
It is necessary to distinguish between logical states, which shall be presented to users, and states in the sense of values of the `status` field stored in the database. The status field of running experiments is periodically updated by Sacred's observers.
In case of an outage or another unexpected event having occurred, the status in the database remains unchanged, for instance in the RUNNING state. Logically separating "dead" experiments from the rest will help in detecting these situations.
For the MongoDB database backend, the fields are mapped from database to the table as described in table 4.2. The field names are relative to the `run` document.
The `result` field has usually a numerical value, but Sacredboard shall raise no error nor crash in case of a different data type in the field.

Logical State	**Colour**	`status` **in DB**	**Additional Condition**
Completed	green	COMPLETED	
Interrupted	orange	INTERRUPTED	
Running	blue	RUNNING	heartbeat time less than 120 seconds ago
Failed	red	FAILED	
Queued	grey	QUEUED	
Dead	black	RUNNING	heartbeat time more than 120 seconds ago
Timed Out	*TBD*	TIMED_OUT	

Table 4.1: Definition of Experiment Run States

Table Column	Database Field
Experiment Name	`experiment.name`
Run State	*see table 4.1*
Start Time	`start_time`
Last Activity	`heartbeat`
Hostname	`host.hostname`
Result	`result`

Table 4.2: MongoDB mapping for the Run Table

F 2 **Sorting Run Table**
Realizes: User Story US 3
By default, the table shall be sorted by the last activity time starting from most recent to the oldest. The user shall have an additional possibility to opt to sort by all the displayed properties in ascending or descending order. The result field should be usually a numerical value, but in case of the MongoDB, the data type solely depends on the experiment source code. Sacredboard shall raise no system error nor crash when sorting different value types.

F 3 **Filtering Runs**
Realizes: User Stories US 4, US 5, US 6
To support browsing the experiment runs, Sacredboard shall allow users to limit the list of displayed experiments to those matching specific criteria. Filtering based on the logical states (see table 4.1) shall be visible and quickly accessible without the need to specify the query manually. Filtering based on other attributes of the experiment run must respect the lack of a fixed database schema – it can only rely on the usual properties that every experiment run has, as documented in section **??**.
The default scope for searching in experiment run is the `config` of the run. However, the user shall not be restricted to that; the filtering mechanism shall enable extending the search area to the whole experiment run.
Logically inappropriate combinations of operators and value types shall be prevented.

F 4 **Viewing Run Configuration**
Realizes: User Story US 8
Sacredboard shall provide a view that displays the run configuration to the user. Run configuration is a dictionary (key-value pairs), where value can be either a primitive object, such as a string or number, an array, or another dictionary. A way of displaying the above mentioned objects must be provided.
For MongoDB, the configuration can be retrieved from the run's `config` property.

F 5 **Viewing Run Extended Information**
Realizes: User Story US 9
Sacredboard shall provide a view that displays the info configuration to the user. The requirement is in many aspects similar to F 4. Additionally, the information may get updated while the experiment is running.
For MongoDB, the extended information can be retrieved from the run's `info` property.

F 6 **Viewing Run's Standard Output**
Realizes: User Story US 10
Sacredboard shall provide a self-updating view that displays the standard output of the experiment run as captured by Sacred.
For MongoDB, the output can be retrieved from the run's `captured_out` field.

F 7 **Launching TensorBoard**
Realizes: User Stories US 11, US 12
Applies to experiments that are using the TensorFlow framework.
Sacredboard shall provide the user with an option to launch TensorBoard for experiment runs that have an associated TensorFlow log directory specified. As there is currently no standardised run property holding the path, it must be defined, and a way of setting it must be proposed. This imposes a new requirement on Sacred (see F 100).
An experiment run can have zero to multiple (probably a few) associated TensorFlow directories. A problem may arise if the log directory is located on a path that is inaccessible from the location where Sacredboard is running on (e.g. it is on another machine). TensorBoard can handle the problem by simply pretending the directory is empty, but the user may not be aware of the issue and can misinterpret the situation.
The interface to TensorBoard is described in section C.

F 8 **Displaying Metrics (Measured Values)**
Realizes: User Story US 13
Sacredboard shall display a chart for each associated metric that gets measured during the execution of the experiment run. Metrics are mostly qualitative variables of the adaptive model, such as the value of the loss function, accuracy of correctly classified items, etcetera. (See section 1.1). Instead of running the training process for a fixed number of iterations, researchers may opt to engage techniques that measure the metrics after each (or e.g. after each 20th) training iteration to decide whether the training process should continue or if it should be terminated because of a low expectation of future progress.
A visual chart of a metric shall be a 2D graph with the training step or elapsed time on the x-axis and the measured scalar value on the y-axis, similarly to the chart in TensorBoard in Figure 4.1. However, as Sacredboard shall only display one chart per metric per run, there shall be only one curve in the graph. A run can have one to a few metrics.

There is currently no standardised way of maintaining the metrics in Sacred. Therefore, a new requirement (F 101) addresses the need.

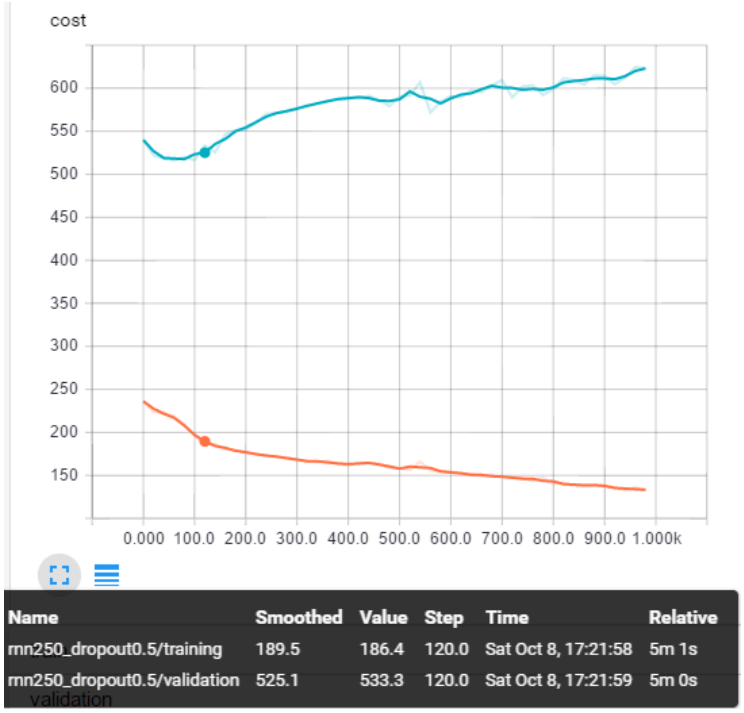


Figure 4.1: TensorBoard chart displaying the classification error evolving over training iterations, measured on training and validation set

F 9 **Deleting Runs**
Realizes: User Story US 16
From time to time, an experiment run may become useless to be maintained in the database, for instance when it fails and provides no interesting information to the researcher. Sacredboard shall provide an option to delete the run.

F 10 **Commenting Experiments**
Realizes: User Story US 16
Sacredboard shall allow adding and viewing additional notes to the experiments so that they can better recall information about the experiment later. As of now, there is no standardised field for comments. For simplicity, a one-to-one relationship (one editable note per run) should satisfy the need for now.

F 11 **Browsing Experiment Source Code**
Realizes: User Story US 15
Sacredboard shall display the experiment source files to the user (read-only).

As the experiment evolves, researchers modify not only hyperparameters of the adaptive model, but also the source files. They should therefore know what version of program actually belongs to which experiment run.
In MongoDB, the files are stored using using the GridFS API (a part of MongoDB) and the corresponding record IDs can be found under the run's `experiment.sources` array as a pair of (`file_name, id`).

Additional Requirements for Sacred

F 100 **Setting TensorFlow Log Directory in Sacred**
System: Sacred
Parent requirement: F 7
Sacred shall define a structure to record directories used to store logs created by TensorFlow during the experiment and shall also provide API to store the information in a convenient way. As of autumn 2016, the new way to create these logs in the TensorFlow API for Python is:
`tf.summary.FileWriter("path/to/logs")`
Although it is a rather unusual use case, a single experiment can produce more than one TensorFlow log per run.

F 101 **Storing Metrics in Sacred**
System: Sacred
Parent requirement: F 8
Sacred shall define a structure to track measurements for different metrics during the experiment run. The values can be recorded either each iteration step or every n-th step, depending on the user's choice. For now, the recorded value is expected to be either decimal or integer number.
It is requested to track each measured value together with the iteration step and the time, because some steps may take longer than others, and when evaluating the model performance, the time needed to train it matters.

4.3.2 Non-functional Requirements

NF 1 **Running on a Local Machine** Sacredboard shall be runnable with a single click or from the command line on the computer that the user is working on, with a database server of the user's choice.

NF 2 **Running on a Server** Sacredboard shall be deployable on a server for an easier access to a shared database by multiple users. This setup is suitable for research groups running their own database server.

NF 3 **MongoDB Backend Support** Sacredboard shall support the Sacredboard MongoDB backend.

NF 4 **Other Backends Support** Sacredboard shall be open to further extensions to support other Sacred backends.

NF 5 **Process Management** *Parent requirement:* F 7
Sacredboard must take care of all processes that it executes (e.g., TensorBoard). Running endless number of TensorBoard instances that nobody uses shall be refrained from.

4.4 Use Cases

This section tries to elaborate the user stories that have a non-trivial scenario and to specify the course of action that the user will take to accommodate his needs.

UC 1 **Filtering Experiment Runs Realizes:** User stories US 4, US 5, US 6 / Requirement F 3

Users	Researcher
Description	Filtering Experiment Runs
Preconditions	Sacredboard is running and the user sees the experiment run list
Base Course of Events	1 The user indicates that the software is to perform filtering of the run list. 2 The software responds by requesting the name of the experiment `config` property to be filtered on, the comparison operator and the searched value. 3 The user inputs the configuration property name and the comparison operator and indicates the type of the searched value (number/string/date). 4 The software applies the selected filter additionally to other filters that have been possibly applied before and indicates that a new filter condition has been added.

Alternative Paths	1 In Step 3, the user indicates he does not wish to perform the search based on an experiment run configuration property but based on another property of the experiment (not necessarily in the run `config` section). The software will allow that. 2 In Step 1, the user indicates that he wants to instantly filter out experiments that have a certain state (e.g. FINISHED and INTERRUPTED). The software must support it without requesting any details about property names. 3 In Step 3, the user indicates he wants to filter based on an invalid combination of operator and value (e.g. number and "contains"). In this case, the software indicates that the choice is invalid and prevents the user from continuing to the next step. 4 In Step 1, the user indicates he wants to remove one of the existing filters instead. The software removes the filter of user's choice. All the postconditions apply.
Postconditions	1 The run list displays only results matching criteria of all the filters applied so far. 2 The software indicates all the filters applied.

UC 2 **Starting TensorBoard**
Realizes: User stories US 11, US 12 / Requirement F 7.

Users	Researcher
Description	Starting TensorBoard dashboard to see advanced information about the run
Preconditions	Sacredboard is running and the user sees the experiment run list
Base Course of Events	1 The user indicates that he wants to launch TensorBoard for some of the experiment runs. 2 The software responds by requesting the user to choose the runs that should be inspected in TensorBoard. 3 The user inputs his choice and confirms it. 4 The software navigates the user to TensorBoard with the desired experiment runs loaded.

Alternative Paths	1 At any time, the user may decide to cancel the action. If so, TensorBoard will not be launched and the user does not get navigated anywhere. (The postcondition does not apply.) 2 If TensorBoard cannot be launched in the last step (for instance, the executable is missing on system `PATH`), a corresponding error message is displayed and the postcondition does not apply.
Postconditions	TensorBoard has started and the user has been navigated to it.

Chapter 5

Design

Having analysed the requirements, they need to be transformed to a functional system. In this chapter, the software architecture of Sacredboard is described accompanied with a short explanation of their relation to the system requirements. The next sections provide basic information about the individual modules and explain the internals of some of them in detail. The rest of the chapter is dedicated to solving design problems that arose during the implementation and that were not considered in earlier stage of the development.

5.1 Architecture

Software architecture deals with the design and implementation of the high-level structure of the software, with abstraction, decomposition and composition, and style and aesthetics. It is the result of assembling architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability. [17]

Defining the software architecture for a particular project is a critical phase, because once stabilized, it cannot be easily changed and most of the future decisions will depend on it. Determining all the necessary processes, components, and interfaces between them only from the information collected during the analysis phase usually results in constructing a basic picture of the desired architecture, but it may take a few individual scenarios to be implemented before the architecture stabilizes. This project was not an exception. The design proposed later in this chapter was originally meant slightly differently than described.

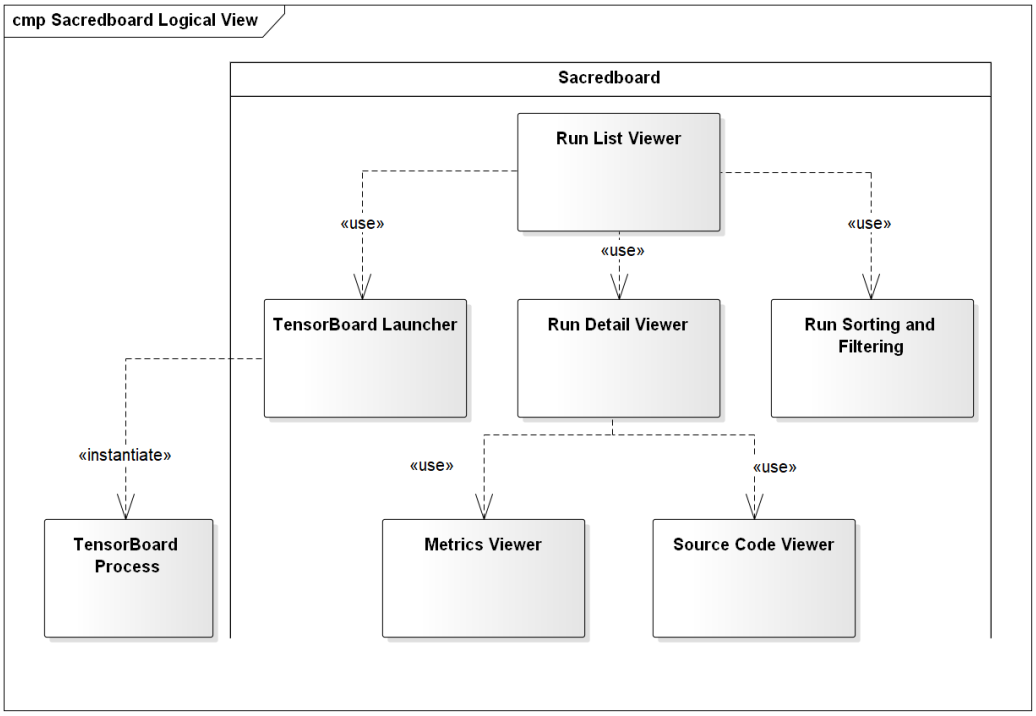


Figure 5.1: Architecture: Logical View

5.1.1 4+1 Architectural View Model

The various aspects of software architecture are difficult to capture in a single diagram, and therefore, Phillippe B. Kruchten, a Canadian software engineer working for Rational Software, developed the 4+1 Architectural View Model in 1995 [17]. To address all the important stakeholder groups of the project, he has proposed decomposing the architecture into four different, yet related views: the logical view, process view, development view, and physical view. There is also the "+1" view, which represents the scenarios: the actual "raison d'être"[9] of the whole project. Different projects may not need to use all of these models, others may need a few more, depending on the concerns of its stakeholders.

Logical View The logical view depicts the key abstractions of the problem domain. It servers the users to show that their needs are addressed in the architecture. Figure 5.1 represents the logical relationships between the functionalities that Sacredboard shall provide to its users. The *TensorBoard Process* is actually not a part of the system, but there is an important relationship that is worth depicting.

[9] the reason for the existence

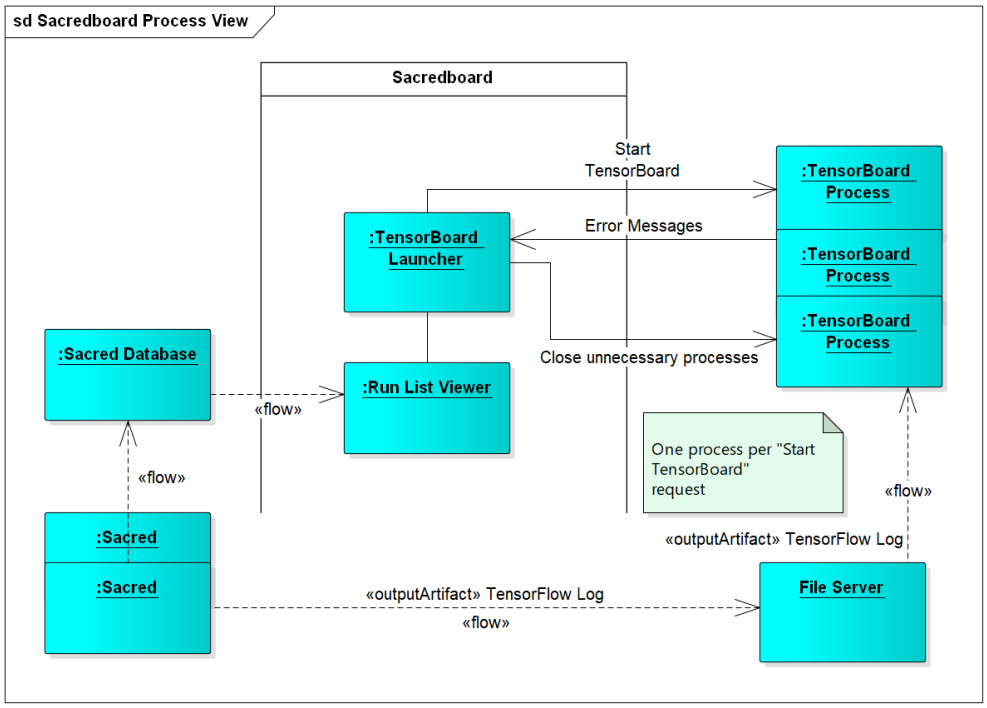


Figure 5.2: Architecture: Process View

Process View The process view, depicted in Figure 5.2, takes into account some non-functional requirements related to system concurrency, distribution, and how they relate to the abstractions from the logical view. The process view can be composed of multiple diagrams, each addressing different concerns on different abstraction levels, such as processes, threads, and scheduled tasks.

The Sacredboard's process view indicates an important architectural concept: the independence of Sacredboard on Sacred's processes. The communication between the two applications will be unidirectional, mediated by the Sacred database backend, which must be running for Sacredboard to work, but no direct communication from Sacred to Sacredboard will ever occur. Sacred processes are independent instances, each running its own experiment. Once an experiment has finished, its process terminates. While running, the information about experiment runs get periodically pushed into the database. Sacredboard must therefore regularly refresh the displayed results too. Similarly, if the experiment uses TensorFlow and produces TensorFlow logs, the TensorBoard process can read them later even if the experiment run has finished. If not, TensorBoard periodically checks the log file for updates.

In contrast, the communication between Sacredboard and TensorBoard will be direct, as Sacredboard will have to start a new process for each user's request to launch TensorBoard, react to the output produced by the external program and detect when the process is unnecessary and terminate it.

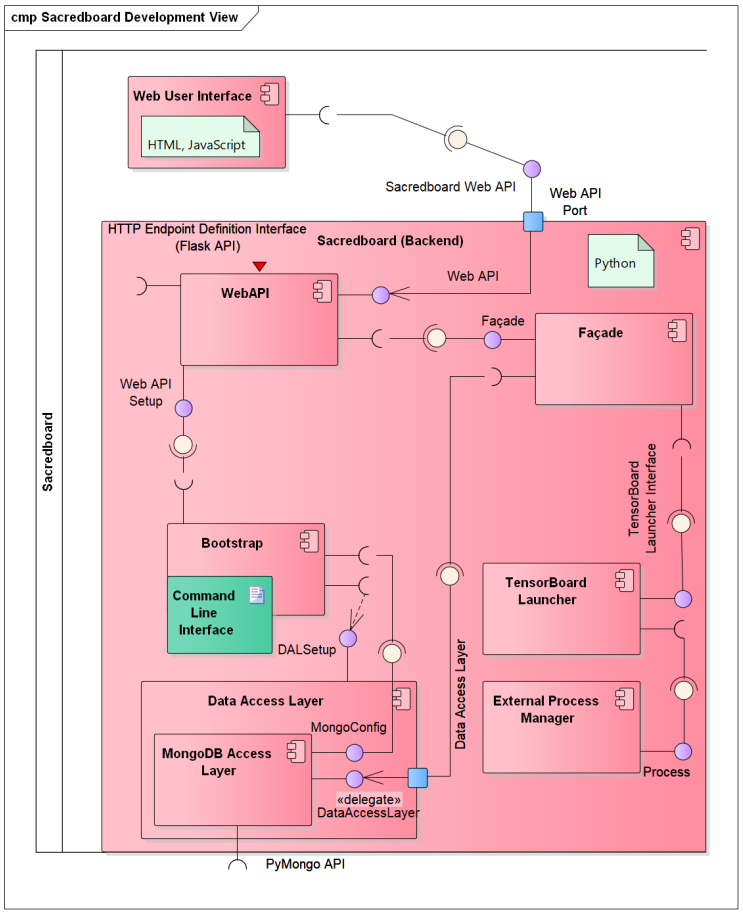


Figure 5.3: Architecture: Development View

Development View The development view focuses on the software module organization in terms of packages or subsystems that can be developed by one or a small number of developers. The subsystems should by organized in a hierarchy of layers, each providing a narrow and well-defined interface to the layer(s) above it. [17]

Figure 5.3 shows the development architecture of Sacredboard, with emphasis on the backend (server) part. The picture defines the modules that need to be developed to meet the functional and non-functional requirements and also already specifies the technology that the software will be built on.

There are two high-level components: the user interface, which will be created as an interactive web frontend, and the backend, which will be responsible for accessing the database and for managing TensorBoard processes. The backend and frontend are described in a more detail in Section 5.3 and 5.4, respectively.

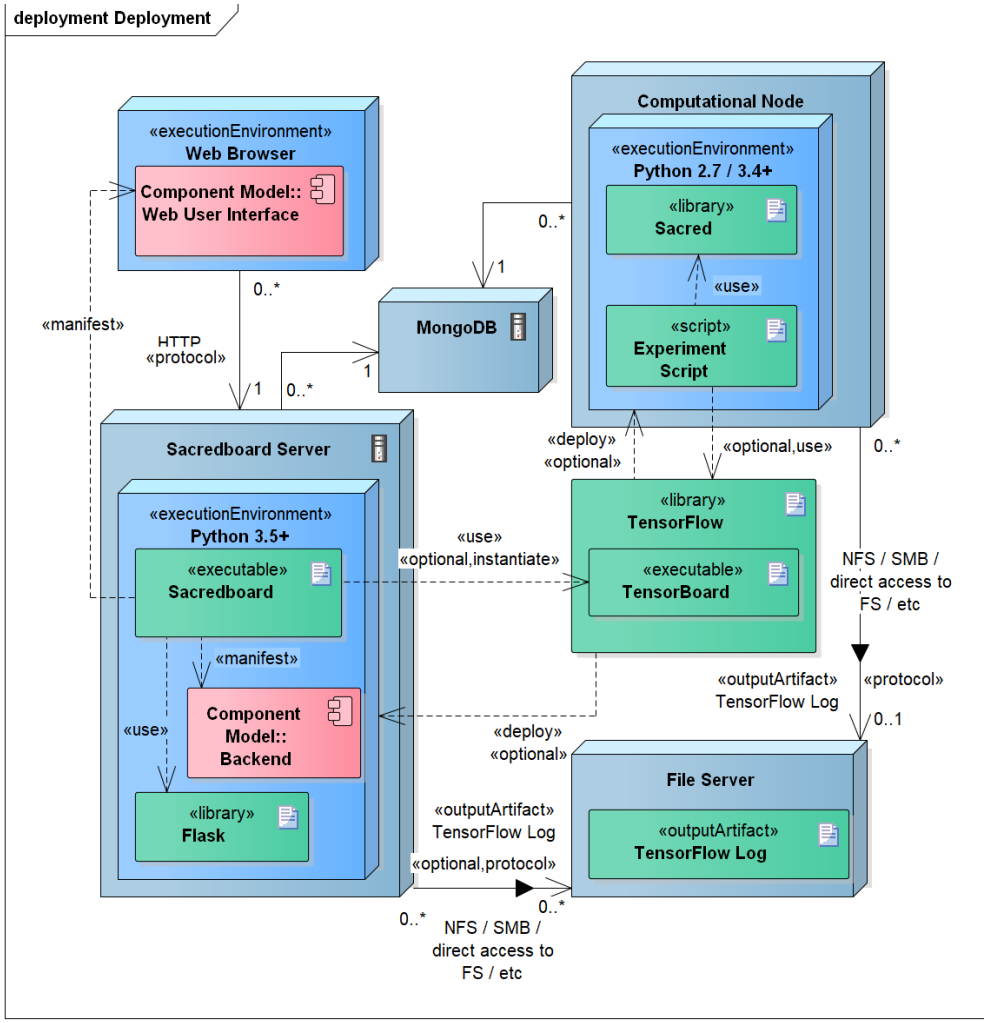


Figure 5.4: Architecture: Deployment View

Physical View The physical view captures the non-functional requirements of the system, such as availability, fault-tolerance, performance and scalability. The nodes on which the software executes are depicted and the various elements identified in the other three views must be mapped onto the nodes. As the software may use different physical configuration for various environments (development, testing, deployment), the source code must not rely on a certain physical layout. [17]

Rather than depicting a concrete instance level physical deployment, there is a specification level deployment diagram provided, as seen in Figure 5.4. The depicted nodes represent logical servers, not physical hardware; the deployment is up to the user provided that certain conditions are met:

- The MongoDB database that is used by Sacred-powered experiments must be reachable from Sacredboard (default port TCP 27017).
- The computer that the user is working with must be able to connect to the Sacredboard server via HTTP (default port TCP 5000). In many cases, users will run the Sacredboard server on their own computers and connect to it from their web browsers locally.
- When using the TensorFlow integration, both the experiment script and Sacredboard must be able to access the produced log files using the same path. It may therefore make sense to think about setting up a file server accessible both from the computational nodes and the Sacredboard server.

5.2 Sacred Extension

Fulfilling requirements F 7 (Launching TensorBoard) and F 8 (Displaying Metrics) requires new features to be developed for Sacred. They are roughly described in requirements F 100 and F 101.

5.2.1 F 100: Setting TensorFlow Log Directory in Sacred

To implement the requirement, two design aspects had to be decided: the data structure that will hold the information and the way how its values will be set by Sacred – the definition of the API to be used by researchers.

Data Structure

Regarding the data structure, Sacred defines a property for storing user-defined information related to the run, called `info`, which is represented as a dictionary and whose content gets periodically serialized and updated in the database when the experiment is running. In case of MongoDB, the mapping from the memory object to the database is one-to-one for primitive types, dictionaries and arrays. This property will be used as the extension point for storing the locations of the TensorFlow log files. The actual names of the files generated by TensorFlow are not important to fulfil the requirement. Because a log can consist of multiple files, all the files must be present in a single directory. The path to that directory is what matters.

The schema will be extended as depicted in Figure 5.5. The `info` dictionary (document in MongoDB) has a key named `tensorflow` whose value embeds another dictionary. The embedded dictionary contains an array of all the paths to the TensorFlow log directories associated with the run. The array is accessible via the `logdirs` key.

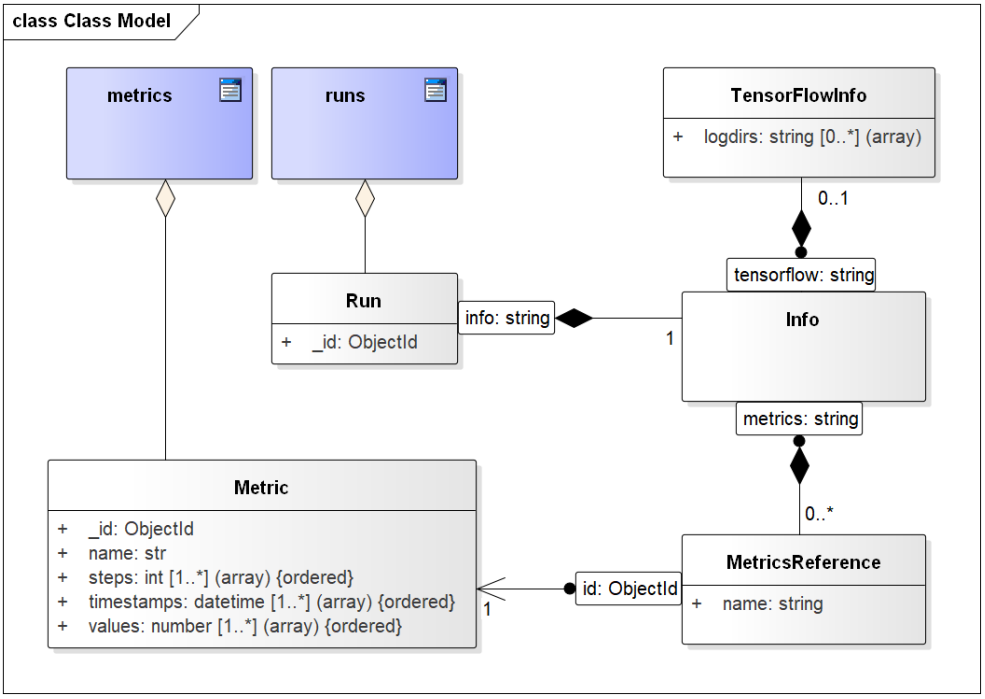


Figure 5.5: Extension of the Sacred Mongo Database Schema

The composition (filled black diamond) arrows represent relations that are essential to the child object; without the connection to the parent object, the child has no meaning. Furthermore, the child object is embedded in the parent as a nested document. The standard association arrow points to a separately stored object, referenced by a key. The aggregation (filled light diamond) arrows point to collections containing the records.

Sacred API

When designing the Sacred API extension that would allow researchers to easily set the log directory path, it was first considered adding a convenience method to the `_run` object that would have to be called each time TensorFlow creates a new log. But in some cases, the logs can be created even without knowledge of the researcher. Other libraries may use TensorFlow as a computational backend in such a way that the researcher-programmer does not interact with TensorFlow directly and it may be the library itself that decides the log directory location. Therefore, instead of explicitly calling a method to append the directory to the Sacred structures, a more flexible solution has been designed to automatically detect the log creation in researcher-definable sections of code.

The researcher may choose to select a portion of his code to be monitored

Listing 5.1: Capturing the Log Directory on a More Granular Level

```
ex = Experiment("my experiment")
def run_experiment(_run):
  with tf.Session() as s:
    with LogFileWriter(ex):
      swr = tf.summary.FileWriter("/tmp/1", s.graph)
      # _run.info["tensorflow"]["logdirs"] == ["/tmp/1"]
      swr3 = tf.summary.FileWriter("./test", s.graph)
      # _run.info["tensorflow"]["logdirs"] == ["/tmp/1",
          "./test"]
    # This is called outside the scope and won't be captured
    swr3 = tf.summary.FileWriter("./nothing", s.graph)
    # Nothing has changed:
    # _run.info["tensorflow"]["logdirs"] == ["/tmp/1", "./test"]
```

for log creation using the *context manager* mechanism as seen in Listing 5.1, or to apply the monitoring on a method level with a *decorator* as in Listing 5.2. The code snippet in the first example tells the program to catch all the calls of the `tf.summary.FileWriter(...)` method within the scope of `with LogFileWriter(ex):`. Any time a new log is created, the path to it gets inserted to the schema as described in Figure 5.5 earlier in this chapter. Similarly, the researcher can indicate his intention to monitor calls from within the scope of a method (or function), including calls from these methods. This is achieved using `LogFileWriter(ex)` as a *decorator* of the function. It is equivalent to wrapping the whole body of the function to the `with` clause.

Both the decorator and context manager approach works by dynamically applying the decorator design pattern to the `FileWriter` method, which gets replaced by a wrapper function that calls the original method and captures the directory parameter that gets inserted into the Sacred structure. After exiting the annotated region or the `with` section, the original method is put back in its place instead of the wrapper.

Making the decorator work is a difficult task due to the necessity of supporting both legacy yet widely used Python 2.7 and the current Python 3.4 and newer. Whilst Python 3 provides API functions to define objects to behave both as context managers and decorators at the same time, this feature is missing in Python 2.7. A workaround had to be found to achieve the same behaviour.

LogFileWriter as Context Manager

Each class implementing the `__enter__` and `__exit__` methods can be used as a decorator. Using the `with` clause can be translated as in Listing 5.3.

Listing 5.2: Capturing the Log Directory on the Method Level

```
from sacred.stflow import LogFileWriter
from sacred import Experiment
import tensorflow as tf

ex = Experiment("my experiment")
@ex.automain
@LogFileWriter(ex)
def run_experiment(_run):
  with tf.Session() as s:
    swr = tf.summary.FileWriter("/tmp/1", s.graph)
    # _run.info["tensorflow"]["logdirs"] == ["/tmp/1"]
    swr2 = tf.summary.FileWriter("./test", s.graph)
    #_run.info["tensorflow"]["logdirs"] == ["/tmp/1", "./test"]
```

Replacement of the `FileWriter` implementation with a custom wrapper and

Listing 5.3: Python Context Manager

```
with LogFileWriter(ex):
  do_something()
# is the same as:
manager = LogFileWriter(ex) # constructor
manager.__enter__()
try:
  do_something()
finally:
  manager.__exit__()
```

back happens in the enter and exit methods, respectively.

LogFileWriter as Decorator

The concept of decorators is a slightly difficult to explain. Essentially, decorator is a callable object (e.g. function) that is put before declaration of another callable object. The decorated object gets replaced by the value returned by the decorator as shown in Listing 5.4.

The example shown is very simple, but instead of returning a primitive value, the `decorator` can return another function (a wrapper) that replaces the decorated function. The decorator receives the decorated function as a parameter that can be passed to the wrapper, which can invoke the original function and perform additional actions to extend its behaviour. In addi-

Listing 5.4: Decorator

```
def decorator(decorated_f):
  return 42

# LEFT: with decorator      # RIGHT: The same without @decorator
@decorator
def decorated():              def decorated():
  print("Hi")                   print("Hi")
  return 100                    return 100
                              decorated = decorator(decorated)
print(decorated)
```

Left: Decorator, Right: The same without @decorator
Outputs: `42`; Notice that `decorated` is no more a function

tion, any Python object is callable provided that it implements the `__call__` method. After many error and trial attempts to make the solution work in all the required Python versions, these properties of Python have been combined in the following manner. Figure 5.6 shows a "simplified" view on the result from the point where the `LogFileWriter(ex)` decorator is used, showing the point where the decorated method is executed and first TensorFlow log is created, to the point where the program exits the monitored region. The new functionality can be found in the `stflow` module of Sacred.

5.2.2 F 101: Storing Metrics in Sacred

Analogously to requirement F 100, both the data structures for Sacred and the API functions had to be proposed. The idea behind maintaining series of measurements is similar to functionality that already available in TensorFlow, but since not every experiment uses the framework, the feature is worth adding.

Data Structure

Each run has zero to several metrics that are distinguished by their name. The choice of the name is fully in competence of the researcher, but for the sake of features that might be added to Sacred(board) in the future, the naming convention should remain consistent. It is advised to name the metrics according to the dataset role and the calculated function, e.g. `training.loss` or `validation.accuracy`.

Figure 5.5 shows the data model of metrics. Each metric consists of a series of trinities `(value, step, timestamp)` organised in three arrays: `values`, `steps` and `timestamps`. As the number of iterations in the run may be large,

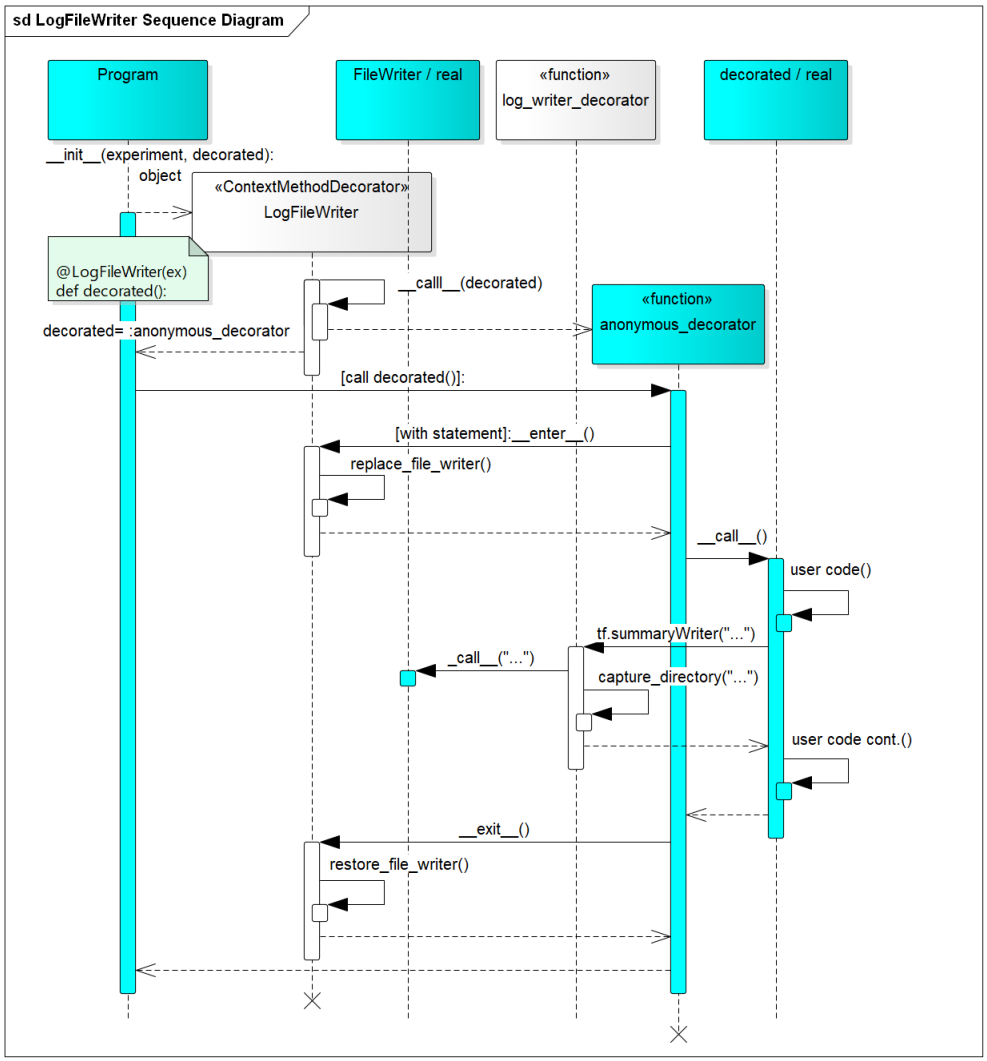


Figure 5.6: Sequence Diagram: LogFileWriter as Decorator

it is often unnecessary to measure the value in every step to get extrapolated results. The `step` is there for the purpose of displaying the measurements on a chart in the correct perspective. The `timestamp`, stored in the UTC zone, serves the same purpose to reflect the time perspective.

In MongoDB, each metric is stored in a separate document in the `metrics` collection and the name and id of the record is referenced to from the run's `info`. The references are organised in an array accessible via the `metrics` key, each item being a dictionary with `id` and `name` properties. The separation prevents the `run` documents from growing with the increasing number of data points being collected, as MongoDB defines a 16 MB per document limit that could be otherwise potentially breached if there was a higher number

of different metrics with lots of data points. Additionally, Sacred rewrites the content of the database run entry each time a heartbeat event occurs, which is normally every 10 seconds, to keep the values in Sacred's runtime memory consistent with the content of the database. Maintaining the metrics in the structure would cause the amount of data being moved to the database increasingly rise after each iteration.

Sacred API

The API extension was proposed to support experiments that use TensorFlow in a seamless way similar to the extension in F 100. A convenient way for experiments not based on TensorFlow should have been provided as well. The integration with TensorFlow had to be postponed due to the high risk of further changes in the TensorFlow API since the framework was still in beta until mid-February 2017. This has been justified as the existing integration for F 100 had to be adjusted after a series of changes accordingly.

The new *Metrics API* is accessible to the running experiments both via the `Experiment` and `Run` objects. The measurements can be added simply by calling the `log_scalar` method that accepts the name of the metric, the measured value and optionally, the step number. Details of usage shown in Listing 5.5.

The recorded data does not get stored to the database immediately. Instead, each call of the `log_scalar` method results in a created message that is pushed to a thread-safe queue (`SacredMQ`). It is retrieved by the `_emit_heartbeat` method of the `Run` class, all the metrics logged since the last heartbeat event are grouped by their name and forwarded to all the Sacred observers used for that run. At the time of writing, only the MongoDB observer is able to process the data. It does so in the `log_metrics(metrics_by_name, info)` method that pushes the newly measured values to MongoDB using its `$push` operator and adds a references them in the `info` dictionary in section "metrics".

SacredMQ

As a part of the implementation, a new mechanism of inter-module communication has been introduced to Sacred. The Sacred Message Queue has been developed to ensure that each Sacred observer receives a copy of the logged values.

In general, when a new message queue is created, a consumer can subscribe to receive messages of that particular queue by calling the `add_consumer()` method. The returned object has a method to check whether there are any unread messages (`has_message()`) and another method to retrieve all the unread messages (`read_all()`). When a new message is published to the queue (`publish(message)`), it gets distributed to all the currently subscribed

Listing 5.5: Sacred Metrics API

```
@ex.automain
def example_metrics(_run):
  counter = 0
  while counter < 20:
    counter+=1
    value = counter
    ms_to_wait = random.randint(5, 5000)
    time.sleep(ms_to_wait/1000)
    # This will add an entry for training.loss metric in every
    # second iteration.
    # The resulting sequence of steps for training.loss
    # will be 0, 2, 4, ...
    if counter % 2 == 0:
      _run.log_scalar("training.loss", value * 1.5, counter)
    # Implicit step counter (0, 1, 2, 3, ...)
    # incremented with each call for training.accuracy:
    _run.log_scalar("training.accuracy", value * 2)
    # Another option is to use the Experiment object
    # The training.diff has its own step counter (0, 1, ...) too
    ex.log_scalar("training.diff", value * 2)
```

listeners. The type of the message does not matter to the queue. Removing a listener has not been implemented yet as it was not needed.

Note: In the end, it was decided that SacredMQ is currently an unnecessary complicated construct for Sacred and will be replaced by a simpler solution.

5.3 Sacredboard Backend

Sacredboard is a client-server application. While the client part is realised as a web-browser frontend, the backend ("server") of Sacredboard is built on Python 3.5 and Flask, a lightweight web framework for Python that comes both with an integrated web server and the capability of deployment on other web servers, such as Apache, using the WSGI[10] interface. The selected solution should help with fulfilling requirements NF 1 and NF 2.

Flask provides Python decorators to design the application URL endpoints in a lucid way and comes with the Jinja2 template system that simplifies rendering of the web pages and resources that the client application is served

[10]Web Server Gateway Interface

with. For accessing the Mongo database, the official PyMongo library has been selected to comply with requirement NF 3.

5.3.1 Backend Modules

The backend is divided into modules as described earlier in Section 5.1.1. The modules are mostly organized in Python packages. As Python prefers "duck typing", the depicted interfaces define rather contracts between the modules in terms of method signature and behaviour than formal interface definitions or abstract classes.

WebAPI

The Web API module uses Flask to expose URL endpoints of the application Web API and serves as an intermediate between the application logic and the frontend client. The Web API uses the concept of resources as in the REST architecture, but perhaps another way of accessing the data in the future could be considered, such as the Graph Query Language (GraphQL) from Facebook. The currently defined endpoints are described in Appendix B.

Data Access Layer

The MongoDB Access Layer is located in the `data` package. The Data Access Layer defines two methods:

- `get_runs(sort_by, sort_direction, start, limit, query)`
  Parameters: All optional
  - `sort_by` The run property to sort by (based on the MongoDB schema), such as "host.hostname".
  - `sort_direction` Either "asc" or "desc". `sort_by` must be specified too.
  - `start, limit` Limit the number of results returned, starting at `start`, ending at `start + limit`.
  - `query` A query to filter the results. The format of the query is described in Appendix B.

  Returns: [a list of runs]
- `get_run(run_id)`
  Return a single run or `None`.

Boostrap

Bootstrap is the entry point for Sacredboard. It takes care of parsing command line parameters, establishes the database connection, starts the integrated web server and opens a new web browser window that navigates the user to the application.

TensorBoard Launcher, External Process Manager

The two modules provide ways of managing external processes, which is currently only TensorBoard, and take care of their correct start and termination. Unfortunately, different operating systems have their means of executing processes and redirecting their standard output to the application. Therefore, the code had to be split for the Windows and Linux operating system to better react for unexpected behaviour of the external program being launched. For instance, Linux supports the *select* mechanism to periodically check whether something has been printed to the standard output by the program. In contrast, Sacredboard has to hope that the program being launched prints something when run under Windows. Otherwise, Sacredboard might wait for the program output forever.

Façade

Façade serves as an intermediate service between the Web API and other modules to separate non-trivial scenarios. The aim is to keep the WebAPI module clean from any logic not related to the exposed web interface so that the functionality of Sacredboard remains reusable even when a different frontend is developed.

5.4 Sacredboard Frontend

The Sacredboard frontend provides the users with an interface to perform all the required tasks. It is built as an HTML and JavaScript application hosted on the Sacredboard backend. The frontend connects to the server part using HTTP to retrieve data and perform operations.

5.4.1 User Interface Design

The user interface of Sacredboard consists of several views. The main screen shows the list of running and finished experiment runs (F 1) in a sortable table (F 2), built using the DataTables [18] extension. In addition to the low fidelity prototype pictured in Figure 5.7, the table also displays the experiment state in a form of a small coloured rectangle. There is a line of textual representations of the experiment states above the table that servers as a legend explaining the state colours. Thanks to checkboxes attached to each of the state descriptions, it is additionally usable for easy filtering on experiment runs based on their state, as requested in F 3 and UC 1, alternative path 2.

For advanced queries, three fields at the top of the page (shown as a single field in the wireframe) are provided for specification of the field name, operator to be used and the filter value. The users immediately sees the scope he is searching in: by default, `config.`. By writing a leading dot (`.`), the user

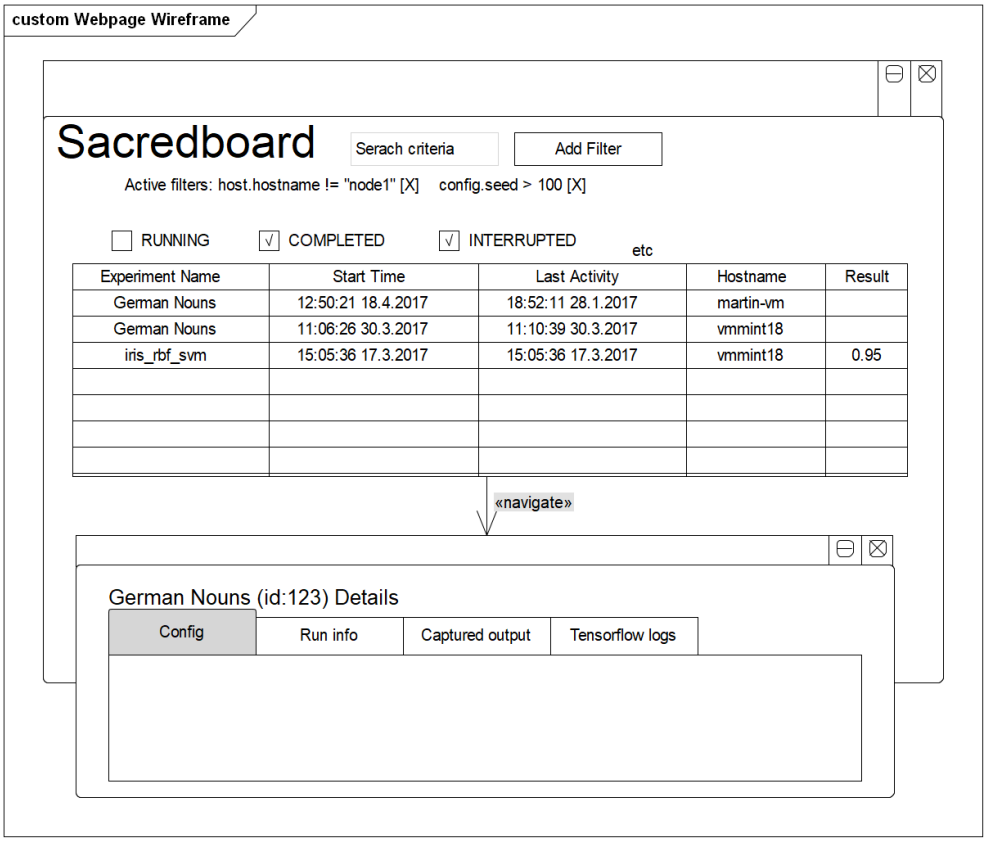


Figure 5.7: Sacredboard Wireframe

indicates the search should be based on any property instead of the default one (alternative path 1 of UC 1). The indicator of searching in `config.` disappears to give the user a feedback to his action. After adding the filter, it appears in a list below from where it can be removed by clicking on its `[X]` control. In case of invalid format or combination of operator and value, the application displays a warning message (figure 5.8).

Clicking on any row in the table opens a collapsible detail view that shows the configuration, extended information, the standard output and buttons for launching TensorBoard for the selected run. This option for launching TensorBoard is, however, not in compliance with UC 2, as it does not support running TensorBoard for multiple runs because of delays in analysis of managing TensorBoard startup for arbitrary combination of runs.

5.4.2 Frontend Architecture Decisions

As the complexity of the frontend grows, managing states of individual components on the page becomes a challenging activity. Developers can easily

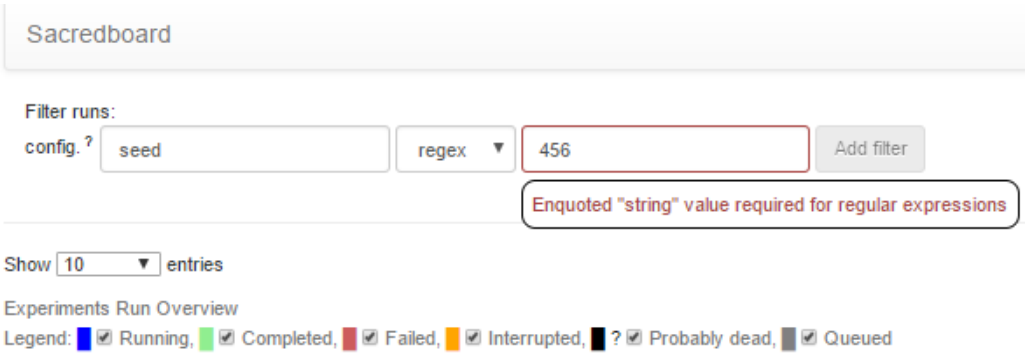


Figure 5.8: Sacredboard Error Message

get lost in the forest of dependencies made of events being triggered on the components and the corresponding actions that shall take place. Additionally, referencing the components directly from the code unnecessarily tightly couples the presentation layer to the model, which significantly reduces reusability and testability.

Frontend developers started realising the problem and since then, a number of libraries addressing the issue has been developed. They solve the decoupling of the model from its view by introducing design patterns already known from the world of desktop applications, such as Model-View-Controller (MVC) or Model-View-View Model (MVVM).

While some of the libraries are rather frameworks enforcing a certain architecture of the whole application frontend to be followed, others can be employed into existing parts of code regardless how the application is built. When deciding the most appropriate solution for Sacredboard, it was necessary to consider the expected frontend complexity: Using no decoupling would work as long as JavaScript is incorporated only for supporting the application interactivity without performing more sophisticated logic.

On the other hand, building the whole frontend on an application framework seems to lead to unnecessary complexity of the release process. Most of the frameworks included in consideration did not work directly with deployable code, but rather relied on their own tools to manage the project and cover various tasks ranging from creating models and components to preparing files that should be finally deployed. Although this approach seems to be a reasonable choice to go for, these tools need to be present on developers' computers, and the developers need to learn how to work with them.

But Sacredboard is expected to be further developed by volunteers recruited from the researchers, and for them, setting up the development environment should be kept preferably simple, especially for those only want to contribute with a small improvement and the intricacy could dishearten them. This has proved to be an adequate requirement, since during writing of the

thesis, the project has been forked by several users who started adding their own features. Therefore, while choosing a suitable library or framework, maintaining a good balance between complexity, features and possible drawbacks of the selected solution must be considered.

5.4.3 Frameworks and Libraries

Before starting development of the Sacred frontend, a number of existing JavaScript frameworks had been studied and judged by the aspects mentioned in Section 5.4.2. It has transpired that most of the considered frameworks were based on a variation of the Model-View-View Model or Model-View-Controller pattern to solve the decoupling problem.

5.4.3.1 Frameworks Studied

The following paragraphs describe a few of the frameworks that were taken into consideration. Some others have not been studied in such detail or have been found at a later stage of the project. All the below mentioned frameworks use the concept of components to represent the view. A component is a web page, a widget, or a single button that can communicate with the controller or model. Components are reusable, they can be instantiated several times on the same page, can be nested, and form a component tree of the view.

AngularJS AngularJS is a predecessor of a more advanced Angular framework. It is built around the data-binding concept, which establishes a connection between the application UI and business logic and further described in Section 5.4.4. Unlike its successor, it does not enforce a particular project structure. The library is simply loaded as any other JavaScript library. AngularJS is currently not under active development, as its developer Google with the community around it now focuses on the new Angular. Therefore, AngularJS has been disqualified for use in Sacredboard.

Angular Framework Even though the Angular framework is a successor to the earlier mentioned AngularJS, it is not backward compatible and except of the name, they do not have much in common. The Angular framework uses the Node.js JavaScript environment for the development process, which provides command line tools for managing the project. Contrary to AngularJS, which was rather a library than a framework, Angular requests the project to maintain a certain structure and takes care of the whole view runtime lifecycle. The initial configuration for the project seemed to be unnecessary complex for a rather smaller project and after discovering the time needed to get familiar with the framework, for the sake of other developers getting involved in the development of Sacredboard in the future, the framework has been rejected too.

Ember.js Similarly to Angular and many other frameworks, Ember.js uses a toolset based on Node.js for managing the development. The view logic is roughly organised in a variation of the Model-View-Controller pattern. Despite the inconvenience of installing Node.js for development, it has been put on a shortlist of frameworks to be further examined, as the configuration overhead was significantly lower than with Angular. Nevertheless, the template engine that renders components in Ember.js used its own syntax rather than enriching HTML with custom notation, which could decrease code readability both for developers and code editors, leading to syntax highlighting and auto-completion not to work properly. It was the main reason to reject the framework.

Aurelia Aurelia is one of the frameworks that has not been come across during the initial phase. Views are built using the Model-View-View Model pattern, components are written in HTML using custom properties. It prefers convention-over-configuration, which reduces the project overhead. The structure and code seems to be minimalistic for simple applications, but the framework guidelines should keep it well-organised even in more complex solutions. It is unfortunate that Aurelia was missed in the beginning, as it would have been examined closer.

5.4.3.2 Selected Solution

Knockout.js Knockout.js belongs rather to the group of libraries than frameworks, as it can be engaged to existing projects without affecting their structure. It decouples the view from model using the Model-View-View model pattern, which is described in Section 5.4.4. Even though it is not the most widely used solution among developers, the following advantages decided the choice: First, Knockout.js itself as well the developed frontend are written in pure JavaScript (and HTML) while keeping coding overhead minimal.

Unlike other solutions, which often utilize syntactic sugar extensions to JavaScript like JSX used in React or TypeScript, Knockout.js requires no translation from such JavaScript superset to the language that browsers understand. In addition, it is left up to the project convention whether new features must use the library or if they rely on a traditional approach of manipulating the HTML document model directly. For Sacredboard, the use of Knockout.js is desirable for components that rely on the view representation – for instance, the developers are encourage to maintain a list of selected items in an array that the view binds to rather than having a method that counts selected elements from the DOM directly.

5.4.4 Model-View-View Model (MVVM)

Model-View-View Model is one of the patterns that separate the responsibility for appearance from the presentation logic. [19] Since it used in Sacredboard, the developers should know about it in order to be able to extend the software further. MVVM is based on three parts: the model, which encapsulates the application logic, the view, which encapsulates the appearance, and the view model, which is responsible for presentation logic and state, as depicted in Figure 5.9.

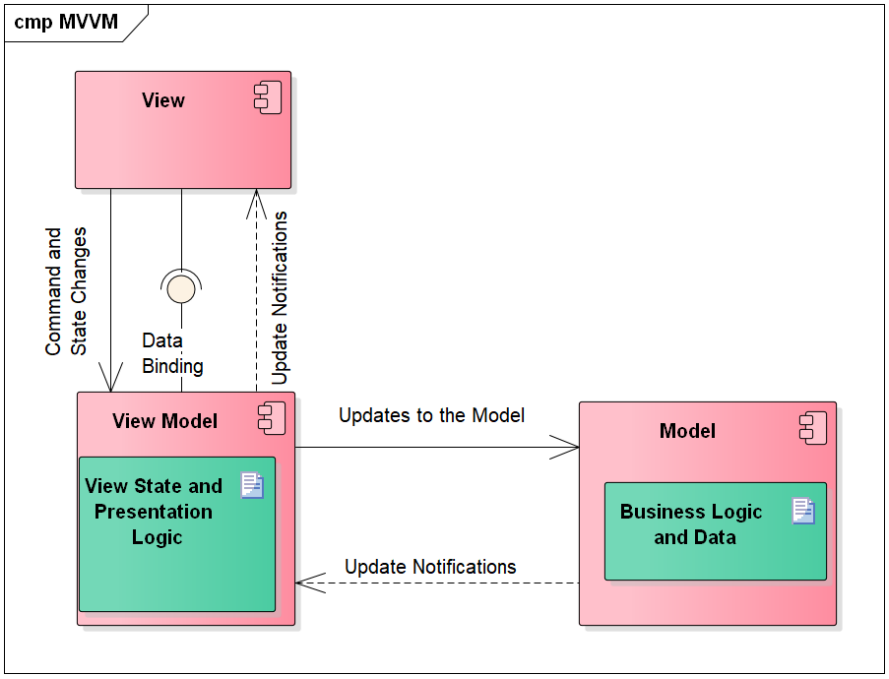


Figure 5.9: Model-View-View Model

View The view is responsible for defining the structure and look of the application. The page or screen that users see is a view per se. It is made of reusable components such as navigation elements, input fields, or canvases for drawing graphics, or a group of other components. In the context of web applications, components are mostly represented by a group of HTML elements. Knockout.js also allows to define own "components" – or rather custom HTML elements – that define both the representation (view), and the backing view-model, as in the case of the `<query-filter>` "component" created in Sacredboard's `filters.js` for realisation of requirement F 3. Each view has to know the *view model* that it represents. That can be either the view model of the parent , or a component-specific view model. The link is realized through data binding, which is described below.

Views should not contain any logic code that needs to be unit tested. Performing tests of the view is typically done via a UI automation testing tool. [20]

Model The objects of the model are responsible for managing the application's data and for ensuring its consistency and validity. [20] It is made of traditional plain old objects that have no relation to the framework standing above them in the presentation layer. They should only expose an interface telling their surroundings about state changes. In terms of client-server architecture, models are responsible for encapsulation of the data that should be moved between the client and the server.

Some sources [21] propose that the model should actually only encapsulate data without any business logic, which should be separated to other classes that act on the model. On the contrary, Martin Fowler, an often-cited software engineer and popularizer of enterprise architecture patterns, has warned about the emergence of so called anemic domain models [22] that lack any logic and serve only as a shell containing data values and relationships. He believes that by extracting logic to separate services, developers are forfeiting the benefits brought by the domain model. But it is necessary to mention he means mainly the models on the application server side, not on the client side. To conclude for the purpose of Sacredboard, logic whose presence in the model can be justified, will be included there.

View Model The view model acts as an intermediary between the view and the model, and is responsible for handling the view logic. Like the controller in Model View Controller, the model view implements commands that are triggered by the users in the view, such as clicking a button or submitting a form, and transforms them into calls to the underlying model.

Additionally, it implements properties which the view can bind to and uses the data from the model to transform them to a format that the view can easily handle. The view model can define computed properties that would not normally be a part of the model, but make the presentation simpler. This can be, for example, joining the given and family name together, converting miles to kilometres, and so on. The view model can also define states of the view that are not directly related to the state of the model: for instance, when an operation is in progress and the view should reflect that by displaying a progress bar, the state of the operation would be also maintained by the view model. [19], [20]

Data Binding

Data binding is the process that establishes a connection between the application UI and business logic. [23] It is a unidirectional coupling of a component to a model, such that the component has a direct reference to a property of

the model but not vice versa. The connection keeps the two parts of the application in sync: Changing the component state updates the bound property and contrariwise, when a change to the model occurs, all the bound views get notified about the modification and automatically update accordingly.

Benefits of MVVM

Individual components that have their own view model can be designed and coded more or less independently. The developers prepare a set of unit tests for the model and another set for the view model. And finally, changing the view does not directly affect the other parts. [19]

5.4.5 Modularity

At the beginning, virtually all the frontend code was put to the main `runs.html` file or was directly being included from there using the `<script>` tag. Not getting lost was becoming more difficult as the development proceeded. Furthermore, to write proper unit tests for the JavaScript frontend, the individual parts of the code should be cohesive, decoupled and independent of the document object model.

There is a commonly used way of managing JavaScript modules: the AMD – Asynchronous Module Definition (AMD) API for JavaScript modules. The AMD only specifies the standards for exposing module interfaces and requiring other modules, but it is not an implementation on its own. Sacredboard comes with the *Require.js* library to take advantage of the AMD. The following modules have been developed: `runs/runTable`, `runs/viewModel`, `runs/filters` and `runs/detailView` to take care of the UI logic.

Chapter 6

Implementation

The term Implementation in the chapter title refers to the process of construction of Sacredboard and the corresponding Sacred extensions. The topic is closely linked to the previous and following chapters about design and testing, as the three "phases" were virtually performed together: although the Sacredboard's software architecture seems to be derived from the requirements, it has emerged not only from them. It had to reflect needs discovered during implementation and testing, as mentioned in the previous section about modularity of the JavaScript frontend. After a preliminary design was created, the implementation of the first prototype could begin.

As the most awaited feature was viewing basic information about experiment runs, a simple HTML table was created to list them. After this stage, the prototype was presented to the main developer of Sacred and his colleagues at IDSIA. The collected feedback was directly incorporated into the code in case of trivial adjustments and new enhancement requests or functional requirements issues were created on Sacredboard's GitHub project page. In the next iteration, the process of choosing the frontend architecture began, which was discussed in Section 5.4.2. Later, Sacredboard was published as a Python package on PyPI (Python Package Index), allowing users to download and install it easily with a single command. New users started adding GitHub issues with their ideas or bugs of their own accord and some have even forked the project to contribute to it.

6.1 Version Control

Sacredboard is an open source project that uses the distributed Git version control system for managing its configuration items. Being hosted [24] on the popular GitHub site, participation of other developers on the project is easy, as any of them is free to contribute to the project by creating their own copy of the official or someone else's repository. The changes made can be either pushed back to the Sacredboard official repository using GitHub's pull requests or

can remain out of the official project. The MIT licensing model[11] allows using Sacredboard and deriving a work from it even for commercial purposes without restrictions or limitations to motivate the developers to continue extending it in the future as they like it.

6.1.1 GitFlow

The Sacredboard repository uses the GitFlow [25] branching model to support code base management. GitFlow is oriented to manage Git branches for development of new features, preparing releases and maintenance. Instead of having the `master` branch to contain the latest features, the GitFlow workflow uses two branches to record the project history: `master` always points to the last release, whereas new features are integrated into the `develop` branch.

Feature branches are based on `develop` and their names follow the `feature/feature-name` convention. Once the development of the feature is finished, it is merged back. When a new version of the software is to be released, a new release branch is created from `develop`, for instance `release/1.1`. This "freezes" the set of features for the upcoming release while enabling the development to continue in its branch. During this period, the release branch gets polished and after that, it gets merged both to `master` and `develop`. A new tag is created to mark the release. For fixes, GitFlow defines hotfix branches (e.g. `hotfix/1.1.1`) that are based on `master` and get merged back in the same way as release branches. There is an exception when there is already a release branch created. In this case, the hotfix should be merged to that release too. When no longer necessary, the extra branches are deleted after a successful merge.

There are command line and GUI tools that support GitFlow and automate the process, even though following the workflow is absolutely possible using the standard `git branch` and `git merge` commands as well. Developer's hands are not tied in any way.

6.2 Development Environment

The entire development of Sacredboard and the Sacred extensions was coded in PyCharm [26] Professional Edition provided under a student license free of charge for educational purposes. Unlike the free Community Edition, the Professional Edition offers support both for Python and Web development, which significantly simplified the development. However, the project itself not tied to any particular IDE.

Sacredboard and the Sacred extension was being developed on Windows using the Anaconda distribution of Python, but later moved first to Linux

[11] The MIT licensed may be later changed to another type of open source license.

and later to the Windows Subsystem for Linux (WSL)[12] for practical reasons – mainly because there was no Windows build for TensorFlow/TensorBoard in the beginning and because of testing of Linux-specific features had to be performed.

6.3 Code and Documentation

The project documentation consists of a Readme file on the Sacredboard's home page, source code comments, this thesis and generated project documentation for users and developers (work-in-progress). The Readme file introduces the project, its features, installation and running instructions, a link to screenshots and basic information about contributing to the project.

The Python source code documentation must follow the PEP 257 Docstring Conventions and the code itself the PEP 8 Style Guide for Python Code. [27] JavaScript code has to comply with the JavaScript *strict mode* and usual conventions, and for each module, at least the exposed interface must be commented using JSDoc comments.

Creation of the developer documentation that will contain instructions on contributing, the Sacredboard architecture, description of modules and code documentation is at the beginning at the time of writing. There is still an open question regarding generating the JavaScript code documentation and integrating it into the documentation for Python code. While many Python projects use the Sphinx Python package to generate an HTML site with documentation, the JavaScript world has its own tools that are mostly based on Node.js, adding another non-python dependency to the project (although only for developers).

Contributions to Sacred have been documented in its user guide, whose sources are a part of the Sacred's GitHub repository.

6.3.1 Project Structure

Developers should know where the individual parts of Sacredboard code reside and the meaning of the configuration files.

- `setup.py` is the central file of the Sacredboard's Python package. It contains information about the package name, version, dependencies and is used to build the package.
- `requirements.txt` lists the runtime Python dependency of Sacredboard. When a new package is necessary to be additionally installed with Sacredboard, its name must be added to the last line of this file.
- `dev-requirements.txt` lists the Python packages necessary for development, documentation generation etc. These must be manually installed

[12] WSL allows running native Linux applications in Windows 10.

by invoking:
`pip install -r dev-requirements.txt`.

- `MANIFEST.in` specifies the list of additional files to be included in the Python package – for instance the HTML and JavaScript files.
- `README.md` contains basic information about the project written in the Markdown syntax.
- `package.json` defines dependencies of Node.js for running automated JavaScript code tests.
- `.eslintrc` defines JavaScript code and documentation style checker.
- `tox.ini` configures `tox` to run all the necessary automated tests.
- `.travis.yml` tells Travis CI to run tests (using **tox** and other tools) when commits are pushed to the GitHub repository.
- `docs` contains source files for documentation (work-in-progress)
- `sacredboard/bootstrap.py` is the entry point file of the application that parses command line arguments and initiates other components
- `sacredboard/app/` is the directory where the backend code is stored.
- `sacredboard/tests/` is reserved for the backend test files.
- `sacredboard/templates/` contains files that need to be processed by the Flask's Jinja2 template engine before serving them to the frontend.
- `sacredboard/static/` is used for various files served to the frontend without being pre-processed by the backend.
- `sacredboard/static/scripts/` contains application-specific JavaScript code for the frontend.
- `sacredboard/static/scripts/tests/` contains the corresponding JavaScript test code.
- `sacredboard/static/vendors/` is used for any third-party frontend dependencies.

6.4 Release Process

Releasing a new version of Sacredboard to the public begins with creating a new release branch from the `develop` branch, e.g. `release/0.2`. All version numbers must be updated to the new version, most importantly the version number in `setup.py`. The Readme file must be updated to contain up-to-date information. After the branch is pushed to the repository, integration tests (of the Python code) are automatically run, including test coverage, code style and code documentation checks. If the build passes all the tests[13], the branch may be merged to both `master` and `develop`.

An additional important step is to publish the version on the Python Package Index (PyPI) site. First of all, the distributable package file must be created:

```
python setup.py sdist
```

[13]The solution for automatic JavaScript tests is still in search.

This creates a new file in the `dist` directory that is going to be uploaded to PyPI using `twine`.

```
twine register dist/sacredboard-0.1.2.tar.gz
twine upload dist/sacredboard-0.1.2.tar.gz
```

Sacredboard is published in the PyPI using credentials belonging to its project administrator (Martin Chovanec), who currently remains responsible for changes and new releases, even though the development itself may be done by the community.

6.4.1 Installation

Sacredboard can be installed on the host computer in several ways. End users will install it using the `pip` package manager:

```
pip install sacredboard
```

The latest development version from the `develop` branch is easy to install too:

```
pip install
    https://github.com/chovanecm/sacredboard/archive/develop.zip
```

Developers may wish to install the package in the development mode that immediately reflects the changes they make. Before doing so, the repository must be cloned from GitHub:

```
git clone https://github.com/chovanecm/sacredboard.git
cd sacredboard
python setup.py develop
```

Instructions how to run the program can be found on the project homepage [24].

Chapter 7

Testing

The Software Engineering Book of Knowledge (SWEBOK [28]) defines testing as a *dynamic* verification that a program provides expected behaviours on a finite set of test cases. The term *dynamic* means that testing always implies executing the program on selected inputs. Software quality management techniques and proofs of correctness are considered to be different from testing. In spite of the definition, the text of this chapter describes both dynamic and static testing of Sacredboard.

7.1 Unit, Integration and Other Tests

Unit testing verifies the functioning in isolation of software elements that are separately testable, whereas integration testing deals with verifying the interactions among software components. [28] Unit tests play a constructive role in test-driven-development when implemented prior to the actual unit code. This practice helps with elaborating requirements of the unit in terms of partitioning the input domain into a collection of equivalent classes and deciding how the unit should react for each of them, including edge cases. Both the Sacred extensions and Sacredboard have been described in this thesis have been mostly developed by writing test cases first. Nevertheless, it is important to mention that passing even well defined test cases does not imply meeting user expectations.

Sacredboard combines Python code with a similar amount of code written in JavaScript. While the first runs on the backend, the latter is executed in the user's web browser. Executing tests of both components requires having either a polyglot test framework or two separate harnesses testing each its part of code. While finding an implementation of the first choice is rather difficult, the second option is feasible when properly configured.

7.1.1 Python Tests

For Python, Sacredboard tests are written on top of the *pytest* [29] framework. To perform regression testing during the development, the fastest way is to do so by executing:

```
python setup.py test
```

Test files are located in the `sacredboard/tests` directory. Some of the "unit" tests written could be rather called "quasi-integration" tests, as they rely on mocking of other components that are not a part of the system.

Quasi-integration tests are used for testing the MongoDB Access Layer and the TensorBoard launcher. Instead of connecting to a real database, its behaviour is imitated by replacing the *pymongo* adapter with the *mongomock* package. This strategy will work as long as behaviour of both the real and mocked components is consistent. The authors of *mongomock* warn of possible differences in unusual use cases.

Similarly, to ensure that the TensorBoard Launcher component correctly handles both expected and unexpected behaviour of TensorBoard, another script was created to imitate the TensorBoard outputs during its start up. The aim was to handle cases when TensorBoard does not start because it is not found, the output does not contain the port that TensorBoard is listening on or if the application simply hangs[14] Unfortunately, and maybe because TensorFlow/TensorBoard is still a new project, the real counterpart sometimes cannot detect when the port it is going to listen on is already in use, which causes problems to Sacredboard.

7.1.2 JavaScript Tests

Writing JavaScript unit tests for Sacredboard was not possible until the code was refactored into testable modules and a suitable test framework was found. As stated in section 5.4.2, it was intended to refrain from using Node.js or any other non-Python system dependency to keep the build process straightforward. Fortunately, there are still pure JavaScript solutions meeting the criteria, but on the other hand, testing the code in the web browser is impractical for automated test execution (see Continuous Integration below).

A compromise solution has been found in the QUnit framework [30] that can run both in the web browser and Node.js. Developers that do not want / cannot install Node.js may run the tests in their web browser by navigating to `http://localhost:5000/_tests` when Sacredboard is running. The other and preferred option is to download Sacredboard's JavaScript dependencies using the Node.js package manager (npm) and run the tests:

```
npm install
# Run JavaScript tests
```

[14] Hang detection works does not work under Windows.

```
npm test
# Run code style check
npm run lint
```

JavaScript tests are located in the `sacredboard/static/scripts/tests` dictionary as modules. To let both the npm and web browser know which tests to run, the corresponding modules must be loaded both in the `tests/index.html` and `tests/node_tests.js` files. Next steps in the testing process for Sacredboard is to select a tool for automated frontend testing that will simulate user interaction with the web page.

7.2 Continuous Integration

To ensure that changes made to Sacredboard do not break the unit tests in the `develop` branch, a task for an continuous integration (CI) server is created every time an update is pushed to the GitHub repository. The build passes only if the following conditions are met:

- Python unit tests pass both Python version 3.4 and 3.5
- JavaScript unit tests pass
- Python code style meets the PEP 8 convention
- Python code documentation is present and meets the PEP 257 convention
- Python code cyclomatic complexity [31] of functions does not exceed 10
- JavaScript code follows defined set of rules
- JavaScript JSDoc code documentation follows defined set of rules

The Travis CI cloud service being used is provided free of charge to open source projects. But developers working on Sacredboard can run the tests even on their local machine. The following additional dependencies must be manually installed to do so:
Python:
`tox`
Other software:
`nodejs >= 4.6`
`npm`

Running `tox` (without parameters) in the project directory launches all the above mentioned tests and informs the user about failures. There are command line options provided to to run only a particular subset of the tests:

- `tox -e py35` – Runs `python setup.py test` with Python 3.5 interpreter. Replacing with `py34` does the same for an older version.
- `tox -e flake8` – Checks the Python code and documentation style, including the cyclomatic complexity of the code.

- `tox -e qunit` – Runs `npm install && npm test` to run JavaScript tests (Node.js necessary).
- `tox -e eslint` – Runs `npm install && npm run lint` to check JavaScript code and documentation style.

7.3 Acceptance Tests

For each of the software requirements it should be possible to validate that the product satisfies it. [28] The plan is to address the users that participated in the questionnaire and let them evaluate the system against the user stories and requirements. Requirements with use case scenarios elaborated should be validated by following all the scenario paths. The tests have not been scheduled yet.

Chapter 8

Conclusion

The aim of the thesis was to support researchers that manage their scientific computing experiments with the Sacred library for Python. Sacred was missing a proper user interface and its users had to rely on their own database queries to access records of the experiments. To solve the issue, the Sacredboard web dashboard has been created as a part of the thesis and Sacred itself has been extended to track new types of information.

Baseline requirements for the user interface and extensions were elicited from the main developer of Sacred and complemented with a survey conducted among users of the current tool. Most of the functional requirements have been implemented while respecting the non-functional requirements that have impacted the software architecture, although some of them should be still considered as experimental. Additional requirements and suggestions for refinements have been received from users after releasing the first version.

The development was accompanied with a series of difficulties that caused delays in the progress. Components depending on third-party libraries did sometimes not behave as they were supposed to, and finding and resolving the root cause of problems required a lot of time or changes to the software architecture. Furthermore, the MVVC framework seemed not to be as flexible as expected and its usage was reconsidered, but in the end, a solution was always found. Another inconvenience were caused by maintaining Sacred compatibility with an old yet widely used Python version 2.7.

Decoupling the Sacredboard frontend code into several modules and using the MVVC pattern contributed to the code readability and comprehensibility. Most of the code is documented (and documenting the rest is to be done) with standardized JSDoc and Python docstrings that will be used to generate a reference guide for future extensions.

The backend code and part of the frontend code is also covered by unit tests that automatically run on the Travis CI continuous integration service. Currently, only the backend and the frontend's model is automatically tested. In spite of its logical decoupling, the web page can break by changing the

HTML (view), improper module loader configuration, a bug in one of the third-party libraries or in integration of the frontend components. Setting up a test suite that checks not only the individual parts but also the web page and its behaviour as a whole is recommended.

Sacredboard version 0.2 is ready to be released. Features for version 0.3 are in development and should be ready before the SciPy 2017 Scientific Computing with Python conference that will take place on July 10–16, 2017 in Austin, Texas. Sacred and its extensions including Sacredboard has been accepted for a presentation as a talk in the Reproducibility track. The underlying paper "Sacred: How I Learned to Stop Worrying and Love the Research" by Klaus Greff (IDSIA / USI / SUPSI), Aaron Klein (University of Freiburg), Martin Chovanec (Czech Technical University) and Jürgen Schmidhuber (IDSIA / SUPSI) is yet to be written.

User Stories – Review

The following user stories have been implemented in version 0.2:
US 1, US 2, US 3, US 4, US 5, US 7, US 8, US 9, US 9, US 10, US 11.
The following user stories are currently in development:
US 13, US 16.
Waiting list:
US 6, US 12, US 14, US 15

Bibliography

[1] Schapire, R. Theoretical Machine Learning (course notes). [online], [cit. 2017-02-27]. Available from: `http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf`

[2] IDSIA. Dalle Molle Institute for Artificial Intelligence. [online], [cit. 2016-09-20]. Available from: `www.idsia.ch`

[3] Greff, K. Sacred. [online], [cit. 2016-09-20]. Available from: `www.github.com/IDSIA/sacred`

[4] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN 0387310738.

[5] USI. Intellignt System (course). [online], [cit. 2017-02-17]. Available from: `search.usi.ch/en/courses/35255538/intelligent-systems`

[6] Schmidhuber, J. Jürgen Schmidhuber's Home Page. [online], [cit. 2017-04-15]. Available from: `http://people.idsia.ch/~juergen/`

[7] Google Inc. TensorFlow. [online], [cit. 2016-10-01]. Available from: `www.tensorflow.org`

[8] USI. Università della Svizzera italiana. [online], [cit. 2016-09-20]. Available from: `http://www.usi.ch/en/`

[9] Schmidhuber, J. Deep Learning Wins MICCAI 2013 Grand Challenge. [online], [cit. 2017-03-14]. Available from: `http://people.idsia.ch/~juergen/deeplearningwinsMICCAIgrandchallenge.html`

[10] Russell, S. J.; Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, second edition, 2003, ISBN 0137903952.

[11] Wikipedia. Artificial Intelligence — Wikipedia, The Free Encyclopedia. [online], [cit. 2017-03-11]. Available from: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=769620718

[12] Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Networks*, volume 61, 2015: pp. 85–117, doi: 10.1016/j.neunet.2014.09.003, published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[13] DARPA. A DARPA Perspective on Artificial Intelligence. [video, online], [cit. 2017-03-21]. Available from: https://www.youtube.com/watch?v=-O01G3tSYpU

[14] Wikipedia. List of datasets for machine learning research — Wikipedia, The Free Encyclopedia. [online], 2017, [cit. 2017-03-14]. Available from: https://en.wikipedia.org/w/index.php?title=List_of_datasets_for_machine_learning_research&oldid=768184217

[15] Greff, K. Sacred – Documentation. [online], [cit. 2017-03-11]. Available from: http://sacred.readthedocs.io

[16] Wikipedia. Iris flower data set — Wikipedia, The Free Encyclopedia. [online], [cit. 2017-03-11]. Available from: https://en.wikipedia.org/w/index.php?title=Iris_flower_data_set&oldid=769153397

[17] Kruchten, P. B. The 4+1 view model of architecture. *IEEE SOFTWARE*, volume 12, 1995: pp. 42–50.

[18] DataTables. [online], [cit. 2017-04-25]. Available from: https://datatables.net/manual/server-side

[19] Microsoft. The MVVM Pattern. [online], [cit. 2017-04-08]. Available from: https://msdn.microsoft.com/en-us/library/hh848246.aspx

[20] Microsoft. Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF. [online], [cit. 2017-04-08]. Available from: https://msdn.microsoft.com/en-us/library/gg405484

[21] Likness, J. Model-View-ViewModel (MVVM) Explained. [online], [cit. 2017-04-09]. Available from: https://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

[22] Fowler, M. Anemic Domain Model. [online], 2003, [cit. 2017-04-09]. Available from: http://www.martinfowler.com/bliki/AnemicDomainModel.html

[23] Microsoft. Data Binding Overview. [online], [cit. 2017-04-08]. Available from: https://msdn.microsoft.com/en-us/library/ms752347

[24] Chovanec, M. Sacredboard. [online], [cit. 2017-04-29]. Available from: http://github.com/chovanecm/sacredboard

[25] Driessen, V. A successful Git branching model. [online], [cit. 2017-04-29]. Available from: http://nvie.com/posts/a-successful-git-branching-model/

[26] JetBrains. PyCharm. [online], [cit. 2017-04-29]. Available from: https://www.jetbrains.com/pycharm/

[27] Goodge, D.; Warsaw, B. Index of Python Enhancement Proposals (PEPs). [online], [cit. 2017-04-29]. Available from: https://www.python.org/dev/peps/

[28] Bourque, P.; Fairley, R. E. (editors). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA: IEEE Computer Society, version 3.0 edition, 2014, ISBN 978-0-7695-5166-1, [cit. 2017-03-23]. Available from: http://www.swebok.org/

[29] Krekel, H. pytest. [online], [cit. 2017-04-29]. Available from: https://docs.pytest.org/en/latest/

[30] The jQuery Foundation. QUnit. [online], [cit. 2017-05-01]. Available from: https://qunitjs.com/

[31] McCabe, T. J. A Complexity Measure. *IEEE Transactions on Software Engineering*, volume 2, 1976: pp. 308–320, [cit. 2017-04-29]. Available from: http://www.literateprogramming.com/mccabe.pdf

[32] Greff, K.; Srivastava, R. K.; et al. Brainstorm: Fast, Flexible and Fun Neural Networks, Version 0.5. 2015. Available from: https://github.com/IDSIA/brainstorm

Appendix A

Sample Project

This chapter shortly presents a simple project that uses the new Sacred features and presents the results in Sacredboard. The mini-project deals with recognizing grammatical gender of German nouns by looking at their textual representation. Before diving into the details, it is worth mentioning that the project was originally using the Brainstorm library [32] from IDSIA for neural network computing. Using the library, relatively good results were observed. For the purpose of this thesis, the project was partially rewritten to TensorFlow. However, due to the lack of experience with building a custom neural network from individual neurons, the results were poor. On the other hand, they are still usable for illustration purposes.

A.1 Problem Description

The vast majority of German nouns belongs to one of three grammatical genders: masculine, feminine, neuter. When a noun appears in a sentence in its singular form, it influences the form of the grammatical article and adjectives standing before it. Therefore, knowing the gender of each noun that the speaker is about to use is necessary in order for the sentence to be built correctly. Unlike some other languages like Italian or Spanish, the noun gender is not always easy to determine by its spoken or written form. Furthermore, the meaning of some words changes according to the gender article used. This is comparable to pronunciation changes in English (*I will read it.* vs. *I have read it.*, *a use case* vs. *Use it!*). For instance, the name *Fulda* refers to a city in the German state of Hessen when used as neuter, but to a river of the same name when used as feminine.

The aim of the mini-research was to find out whether a certain number of nouns learned with their correct gender helps with guessing the gender of other nouns. The test assumes that non-native speakers have a relatively large German vocabulary, but remember only a smaller subset of nouns with their correct gender. Real learners may already known some rules that help

with deciding gender of the other nouns, for example that joining two or more words together causes the resulting word to adopt the gender of the last word in the row (*die Bahn* (the railway) + *der Hof* (the court) = *der Bahnhof* = the railway station). But these rules are not explained to the algorithm.

A.2 Data Preparation

The word data set consisted of 2080 German nouns and their article determining the gender. Most of the words were gathered from different web pages and merged together. For the use in the experiment, the words were stored as a 2D matrix of numbers: one dimension indexed the word and the other character codes of the individual letters.

A.3 Recurrent Neural Network with TensorFlow

The experiment was using a recurrent neural network (Figure A.1). Input vectors of words (x) were fed to the network in small batches, each character encoded as a hot-one vector[15] that activates one of the neurons in the input layer. The number of neurons in the recurrent network is controlled by Sacred in its configuration parameter `hidden_size`. After the recurrent network receives the whole word, some of the outputs of the neurons are randomly deactivated (their value is set to zero) to prevent overfitting – a situation that leads to a model that is "overtrained" and instead of remembering patterns, it learns the training words "by heart" and fails for other inputs. The probability of a neuron not being dropped is defined as the `dropout_keep_probability` Sacred configuration parameter.

The output of the dropout layer are 500 values that "vote" for the classification result. There is a softmax 'committee" layer that uses weighted sums to compute the probability of the word belonging to each of the three classes. After the result is compared with the target (correct gender) for all the words, the overall accuracy of the model is computed, accompanied with the mean cross entropy that servers as the error function that the training process should minimize.

A method of stochastic gradient descent (Adam, Adaptive Moment Estimation) is used to compute gradients of network's internal weights based on the error function. The Adam optimizer follows the computed gradients to minimize the cross entropy and make the network present desired results. Unfortunately, for a reason that has not been discovered until the deadline of the thesis, the network seems to be unable to generalize on words in the test set.

[15] vector of many 0 and a single 1

The code that defines the network and runs the training process is available on the media attached to this thesis in file `train_model.py`. A snippet of this file is also presented in Listing A.1.

A.4 Observing with Sacredboard

The experiment progress was observed using Sacredboard. Figure A.2 shows experiment runs containing "German" in their name that were training a neural network of at least 250 neurons in the recurrent layer. Figure A.3 displays the configuration parameters of one of the runs and in Figure A.4, there is a plot of the cross entropy error function for one of the runs. The metrics plot uses the Plot.ly library and is in the stage of prototyping at the time of writing.

As declared in the beginning of this chapter, the mini-project with TensorFlow was not successful. Furthermore, tutorials found on the Internet were sometimes not updated to reflect changes in beta versions of TensorFlow, which slowed down the development. Now, when the API has stabilized, the results could be improved if more time is invested into the problem.

Nevertheless, for those who are curious about the results: using the above mentioned Brainstorm library and a LSTM (Long short-term memory) network with 1000 units, it was possible to correctly classify 63,2 % of unknown nouns after 5 000 of training iterations on 347 words. When using a technique different from neural networks, the decision tree, and limiting the training data to last three letters of each word, the model could guess 67 % nouns correctly. As mentioned, there are certain rules that German learners explicitly learn. Therefore, it is likely that it is possible to create a "sense" for guessing the correct article to use, provided that the user of the language has already a certain level of knowledge.

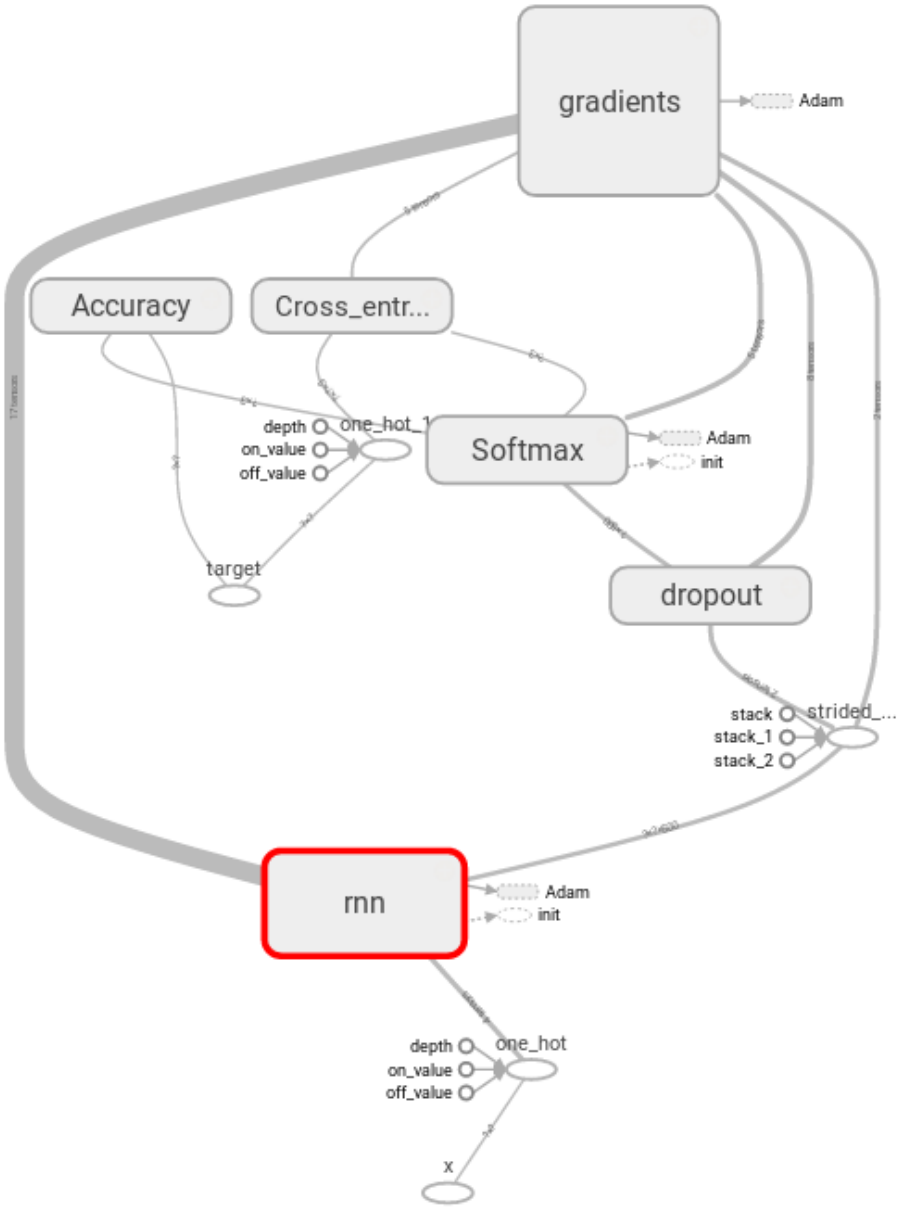


Figure A.1: German Noun Classification Network (Visualisation in TensorBoard)

Listing A.1: Tranining Neural Network – train_model.py

```
@ex.automain
@LogFileWriter(ex) # Save pat to TensorFlow log to
    Sacred DB
def runExperiment(...):
classifier = construct_model()
(...)
# Create TensorFlow log file
writer = tf.summary.FileWriter(log_dir, sess.graph)
(...)
while i < total_training_steps:
# calculate accuracy and cost function for training data
tr_summary, tr_accuracy, tr_cost =
sess.run((train_sum_op, classifier.accuracy,
    classifier.cost),
feed_dict={classifier.inputs: inputs,
    classifier.targets: targets})
# save it to TensorFlow log
writer.add_summary(tr_summary, i)
# the same for validation data
va_summary, va_accuracy, va_cost =
sess.run((validation_sum_op, classifier.accuracy,
    classifier.cost),
feed_dict={classifier.inputs: va_inputs,
    classifier.targets: va_targets})
writer.add_summary(va_summary, i)
# Use the new Sacred Metrics API to store the data to
    the Database
_run.log_scalar("training.accuracy",
    float(tr_accuracy), i)
_run.log_scalar("training.cost", float(tr_cost), i)
_run.log_scalar("validation.accuracy",
    float(va_accuracy), i)
_run.log_scalar("validation.cost", float(va_cost), i)
(...)
classifier.train(sess, inputs_batch, targets_batch)
i+=1
```

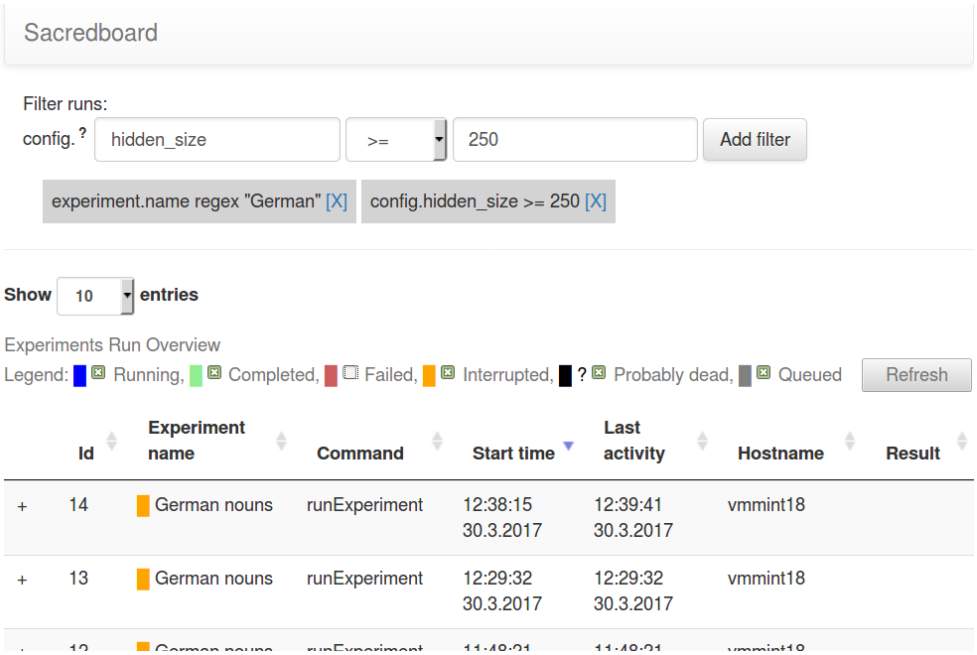


Figure A.2: Sacredboard: List of Runs

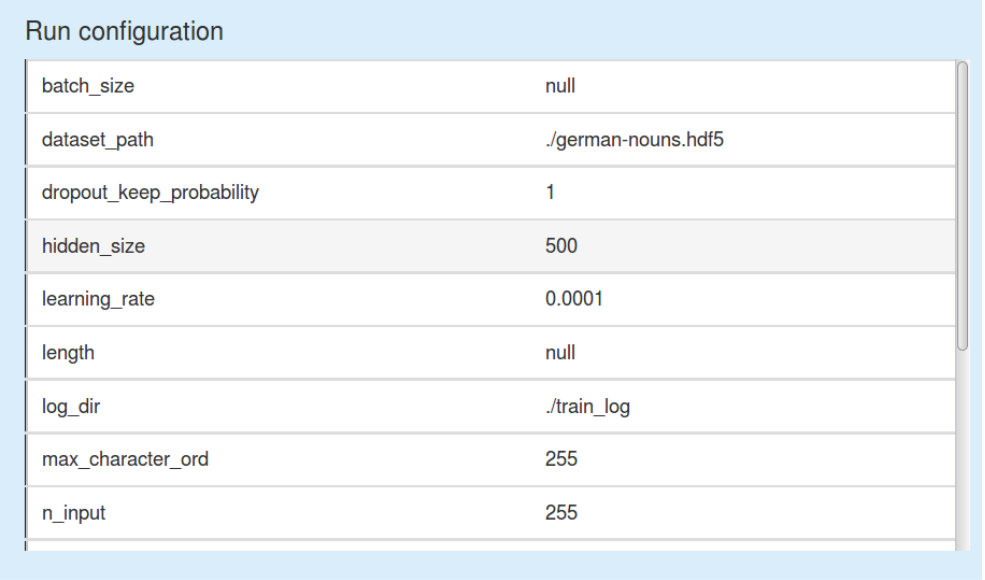


Figure A.3: Sacredboard: Interactive Configuration Browser

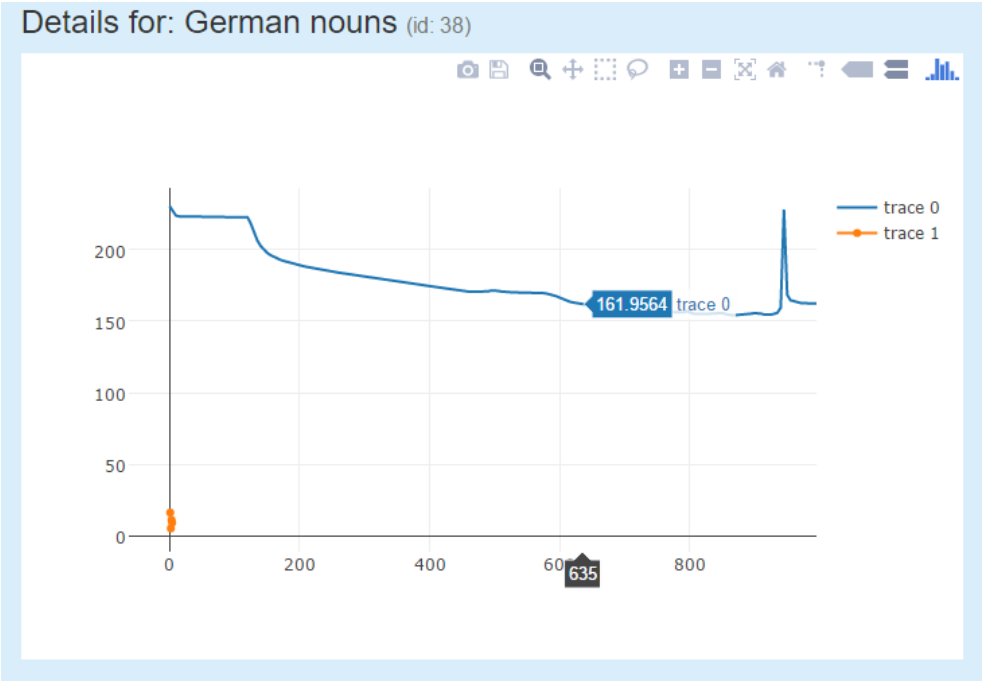


Figure A.4: Sacredboard: Mean Cross Entropy Metrics Plot (hi-fi prototype)

Appendix B

Sacredboard HTTP Interface Specification

- `/`, `/runs` (GET) – **Frontend Page**
  Return Type: HTML
  Serves the front HTML page that displays the main view and requests data from other endpoints. Additionally, the implicit `/static` endpoint points to a directory of the same name to provide JavaScript, CSS and other resources.

- `/_tests` (GET) - **Frontend Unit Test Page**
  Return Type: HTML
  Points to the test page that runs unit tests of the frontend.

- `/api/run` (GET) - **List of Runs**
  Return Type: JSON
  Realizes: F 1, F 2, F 3
  Returns a list of experiment runs in a format required by DataTables [18]. The structure of the `data` elements is as in Table B.1

Field	Definition	format
`id`	experiment run id (from the DB)	string or number
`status`	status (see F 1)	string
`is_alive`	updated in last 120 sec	boolean
`start_time`	run start date and time	string, based on locales
`heartbeat`	run last activity date and time	string, based on locales
`heartbeat_diff`	seconds since last update	double
`hostname`	the node the experiment runs on	string
`result`	Result of the experiment	number or string

Table B.1: Web API Run Resource

The returned results can be filtered using a JSON object passed as a string via the `queryFilter` URL parameter:

```
{"type":"and","filters":[
  {"type":"or","filters":[
    {"field":"status","operator":"==","value":"FAILED"},
    {"field":"status","operator":"==","value":"DEAD"}]},
  {"field":"host.hostname","operator":"==","value":"NTBACER"}]}
```

The parameter is built from clauses. Each clause is either conjunctive (`"and"`), disjunctive (`"or"`), or a "terminal clause". Each of the the earlier two types must specify the `filters` array of other clauses to be joined together by that logical connective. A terminal clause does not specifies its type, instead, it has a different set of fields: the `field` to be queried on (based on the MongoDB schema, using dot notation to access nested documents), the `operator` (one of `"=="`, `"!="`, `"<"`, `"<="`, `">"`, `">="`, and `"regex"` for regular expressions. The `value` field contains the value to be compared with (either a string or a number). Notice that for the `"status"` field, the `RUNNING` and `DEAD` runs are distinguished by the backend, even though the corresponding database field does not distinguish these two states.

- `/api/run/<run_id>` (GET) – **Get Run**
  Return Type: JSON
  Realizes: F 4, F 5, F 6
  Returns a single run based on the ID. The recorded structure is the same as for **/api/run** with a few exceptions: Either exactly one item is returned or the HTTP 404 error occurs if the run was not found. Additionally, the returned object embeds the full MongoDB document transformed to the JSON format. It is accessible via the `object` field.

- `/tensorboard/start/<run_id>` (GET) - **Start TensorBoard Realizes:** F 7
  Starts TensorBoard for the given run ID and TensorFlow log index. The log index is the index to the `info.tensorflow.logdirs` array, which contains paths to the log directories associated with that run. **Returns:** HTTP redirect to TensorBoard or an error code and message
  Known issues: TensorBoard seems failing to detect when the port is in use on some system configurations. It pretends to be listening while it is not. As a result, the user may be redirected to another TensorBoard instance. A workaround has to be found, such as letting the user pick the port.

- `/tensorboard/stop` (GET, POST) - **Stop TensorBoard**
  Realizes: NF 5
  Stops all TensorBoard processes started by Sacredboard.

Appendix C

TensorBoard Interface

TensorBoard can be launched by executing the tensorboard script `tensorboard --logdir LOGDIR`. It comes with an integrated webserver listening by default on port 6006. The port number that the server is listening on is printed to the standard output during startup. Another port can be specified by the `--port PORT` option.

The `LOGDIR` option is mandatory and refers to one or more TensorFlow log directories. The directories are searched recursively for log files, which are then displayed in layers on rendered charts. The same behaviour, even if the directories are in different directory trees, can be achieved by joining them by a comma (,), e.g. `tensorboard --logdir=alias1:path1,alias2:path2`. Aliases help the user to distinguish between the directories when using TensorBoard.

The `tensorboard` script is known to print one of the following messages to the standard output after being run with the `logdir` option:

- `Starting TensorBoard b'41' on port 6006` (Linux)
  `Starting TensorBoard b'39' on port 6006` (Windows)
  - Meaning under Linux: TensorBoard has successfully started.
  - Meaning under Windows: TensorBoard has either started or the port was in use. In the latter case, there is probably no reasonable way of detecting the issue. ...
- `Tried to connect to port 6006, but address is in use.` (Linux only)

Note: The port numbers in the ouput may vary.

Appendix D

Acronyms

AI Artificial Intelligence
AMD Asynchronous Module Definition
API Application Programming Interface
DNN Deep Neural Network
DOM Document Object Model
F Feature / Functional Requirement
GPU Graphics Processing Unit
GUI Graphical User Interface
hi-fi High-Fidelity
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
IDE Integrated Development Environment
IDSIA Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
ML Machine Learning
MVC Model-View-Controller
MVVM Model-View-View Model
NF Non-functional Requirement
PEP Python Enhancement Proposals
PyPI Python Package Index
RNN Recurrent Neural Network
SQL Structured Query Language
SUPSI Scuola universitaria professionale della Svizzera italiana
SVM Support Vector Machine
SWEBOK Software Engineering Book of Knowledge
TBD To Be Determined
UC Use Case
UI User Interface
URL Uniform Resource Locator
US User Story
USI Università della Svizzera italiana

UTC Coordinated Universal Time
WSGI Web Server Gateway Interface

Appendix E

Contents of the Enclosed Media

- `readme.txt` ..... Content description
- `src` ..... the directory of source codes
  - `sacredboard.zip` ..... Sacredboard source files (develop branch)
  - `sacred.zip` ..... Sacred source files (feature/data-logger branch)
  - `example.zip` ..... an example experiment using Sacred
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `EAP` ..... Enterprise Architect Project Directory
- `text` ..... the thesis text directory
  - `DP_Chovanec_Martin_2017.pdf` ..... the thesis text in PDF format