Universiteit Leiden

# Master Computer Science

GreatAI: An easy-to-adopt framework for robust end-to-end AI deployments


| | |
|---|---|
| Name: | András Schmelczer |
| Student ID: | s3052249 |
| Date: | 03/08/2022 |
| Specialisation: | Advanced Computing and Systems |
| 1st supervisor: | Prof. dr. ir. Joost Visser |
| 2nd supervisor: | Dr. Suzan Verberne |


Master's Thesis in Computer Science


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

**Abstract**

**Background:** Despite its long-standing history, artificial intelligence (AI) has only recently started enjoying widespread industry awareness and adoption; partly thanks to the prevalence of frameworks accessibly exposing state-of-the-art models. In order to achieve robust production deployments, the successful integration of AI components demands strong engineering methods. Concerningly, a tendency seems to be unfolding: even though professionals already have access to frameworks for deploying AI correctly, case studies and developer surveys have found that many deployments do not follow best practices.

**Objective:** This thesis sets out to investigate the reasons behind the asymmetry between the adoption of accessible AI libraries and existing reusable solutions to robust deployments. A software framework called *GreatAI* is designed which aims to facilitate General Robust End-to-end Automated Trustworthy AI deployments while attempting to overcome the practical drawbacks of its predecessors.

**Method:** The utility of *GreatAI* is validated by applying the principles of design science methodology through iteratively designing it in a case study of a commercial text mining pipeline. Subsequently, interviews are conducted with practitioners for validating the generalisability of the design.

**Results:** To do.

**Conclusions:** To do.

# Contents

# Acknowledgements

I wish to extend my special thanks to my supervisors, Prof. dr. ir. Joost Visser and Dr. Suzan Verberne, for their invaluable assistance and guidance. I would also like to thank ScoutinScience B.V. for our fruitful collaboration and for allowing me to use their software for experimentation, validation, and demonstration purposes. Last but not least, I would like to individually thank Name, Name, ..., and Name for participating and providing insightful feedback in our interviews.

# Chapter 1

# Introduction

Artificial intelligence techniques have recently started enjoying widespread industry awareness and adoption; the use of AI is increasingly prevalent in all sectors [1, 2]. The reasons behind this are manifold [3], to name a few: recent breakthroughs in deep learning (DL), increased public awareness, abundance of available data, access to powerful low-cost commodity hardware, education, but most interestingly, the rise of high-level libraries making ready-to-use state-of-the-art (SOTA) models easily available. The latter practically abolishes the barrier of entry for applying AI — and with that — can help use-cases in various areas.

However, the successful integration of AI components into production-ready applications demands strong engineering methods in order to achieve robust deployments [4]. That is why it is as important as ever to also focus on the quality and robustness of deployed models and software. For instance, the lack of a proper overview of data transformation steps may lead to suboptimal performance and to introducing unintended biases which might contribute to the ever-increasing negative externality of misused AI [5].

Concerningly, a peculiar tendency seems to be unfolding: even though industry professionals already have access to numerous frameworks for deploying AI correctly and responsibly, case studies and developer surveys have found that a considerable fraction of deployments do not follow best practices [4, 6, 7, 8, 9]. Utilising state-of-the-art machine learning (ML) models has become reasonably simple; applying them properly is as difficult and nuanced as ever.

This thesis is set out to investigate the reasons behind the apparent asymmetry between industry adoption of accessible AI-libraries and existing reusable solutions for robust AI deployments. It is hypothesised that the primary reason for the underwhelming adoption rate of best practices is the short supply of professionals equally proficient in the domains of both data science and software engineering. Nevertheless, even without their presence, practitioners could rely on frameworks for automated mature deployment processes. However, the barrier of entry for using such existing libraries is too high, especially when compared with the simplicity of AI-libraries.

Therefore, a software framework — called *GreatAI*[1] — is designed and its design is presented in this thesis. The principal motivation behind the construction of *GreatAI* is to facilitate the responsible and robust deployment of algorithms and models by designing an accessible API in an attempt to overcome the practical drawbacks of other, similar frameworks. Its name stands for its main aim: to assist easily creating <u>G</u>eneral <u>R</u>obust <u>E</u>nd-to-end <u>A</u>utomated, and <u>T</u>rustworthy AI deployments.

[1] github.com/schmelczer/great-ai

The utility of *GreatAI* is validated using the principles of design science methodology [10] through iteratively designing its API and implementation in a case study concerning the text mining pipeline for a commercial product in collaboration with ScoutinScience B.V.[2] The goal of the aforementioned software suite is to evaluate technical transfer opportunities in scientific publications. Subsequently, interviews are conducted with practitioners for validating the generalisability of the design.

## 1.1 Research questions

I hypothesise that facilitating the adoption of AI deployment best practices is viable by finding less complex framework designs which are easier to adopt in order to decrease the negative externality of misused AI. This paper sets out to investigate the hypothesis by answering the following research questions.

**RQ1.** To what extent does the complexity of deploying AI hinders industrial applications?

**RQ2.** What API design techniques can be effectively applied in order to decrease the complexity of correctly deploying AI services?

**RQ3.** To what extent can *GreatAI* automatically implement AI deployment best practices?

**RQ4.** How suitable is the design of *GreatAI* for helping to apply best practices in other contexts?

In this case, complexity refers to the difficulty faced by professionals (Data Scientists and Software Engineers alike) when integrating third-party libraries with their solutions. This could be also described as the barrier of entry or steepness of the learning curve. If the aforementioned hypothesis is correct, the adoption of best practices can be efficiently increased by decreasing this complexity. AI deployment best practices entail the technical steps ought to be taken in order to achieve robust, end-to-end, automated, and trustworthy deployments. These are detailed in Section 4.2.

The existence question regarding the problem itself (**RQ1**) is answered by reviewing the literature of more than 30 published case studies in Chapter 2. **RQ2** and **RQ3** are closely connected, the design and evaluation phases utilised to answer them follow an iterative process. They are examined in Chapters 4 and 5 respectively. The final evaluation step is to ascertain the capability of the framework's design to generalise beyond a single subdomain and problem context. This question, **RQ4**, is investigated through interviews with industry professionals in Chapter 6.1.

## 1.2 Structure

The rest of the thesis is organised as follows: Chapter 2 approaches the problem and the state-of-the-art from three perspectives: the recent trends of AI-library API designs, the experiences gained from practical applications, and a comparison of existing deployment options. Next, the methodology utilised for the subsequent chapters is described in Chapter 3. The design cycle is broken into two chapters, Chapter 4 and 5. The former clarifies the scope and describes the design principles, while the latter details the specifics of the practical case study, the framework's interaction with it. The contributions of the novel design and obtained results are shown and further validated by conducting interviews with industry professionals in Chapter 6. The thesis is concluded in Chapter 7.

[2]scoutinscience.com

# Chapter 2

# Background

Despite the long-standing history of artificial intelligence, industry awareness and adoption has only recently started to meaningfully catch up [1]. At the same time, more regulations and guidelines are being published, for instance, the Ethics guidelines for trustworthy AI by the European Commission's High-Level Expert Group on AI[1]. This contains seven key requirements, including human agency and oversight, technical robustness, safety, transparency, and accountability. When it comes to accountability, there are clear advances being made [11], however, in the case of the other requirements, the situation is more nuanced. Thankfully, the field of software engineering for machine learning (SE4ML)[2] has been working towards finding ways to assist data scientists and software engineers in ensuring these (and more) expectations are met by their software.

In the following, the context of the problem is presented from three perspectives. Starting with its possible cause: the democratisation of state-of-the-art AI algorithms and models. Subsequently, the challenges encountered when applying AI in practice are outlined by case studies and survey data. Lastly, the existing approaches and solutions are introduced.

## 2.1 Accessible AI

Most companies prefer not to develop new models but instead reuse prior ones [2] and they are able to do so increasingly easier. In recent years, there has been a proliferation of highly accessible AI libraries. For example, let us consider the domain of natural language processing (NLP). There are various options for finding AI solutions that work out of the box: FLAIR [12] and Huggingface's transformers [13] let developers access the state-of-the-art models and methods in only a couple of lines of code (in many cases 2 or 3). Using transfer-learning, Huggingface enables developers to leverage vast amounts of knowledge learned by pretrained models (such as BERT [14] and its many improved variations) and fine-tune them for their specific use-case. The API exposing this is also extremely accessible.

It is not just these two packages, the list of readily available tools is vast: SpaCy [15], Gensim [16], and scikit-learn [17], XGBoost [18] are other great examples. The situation is similar in all subdomains of artificial intelligence: some domain expertise is — admittedly — beneficial but not a hard-requirement.

[1] digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[2] Both in practice and in the literature, this is sometimes also referred to as *AI Engineering* and has a large intersection with — or arguably is the same as — *MLOps*.

This, combined with the exponentially increasing computing power affordably available to consumers and business alike [19], results in AI that is accessible by many.

## 2.2 State of the industry

In contrast to this trend, the software landscape around packaging, deploying, and maintaining machine learning (ML) — and in general — data-heavy applications paints a different picture. Fortunately, the related issues and their ramifications have been already thoroughly investigated.

When looking at AI/ML[3] code in practice through the lens of technical debt, Sculley et al. [9] emphasise the repercussions of writing *glue code* between the algorithms and different systems or libraries and define it as an anti-pattern. The consequence of this is the advice against using generic libraries because their rigid APIs may inhibit improvements, cause lock-in, and result in large amounts of glue code. This is a recurring theme in discussions with industry professionals.

Haakman et al. [6] interviewed 17 people at ING which is a well-known fintech company undergoing a digital transformation to embrace AI. They found that the existing tools for ML do not meet the particularities of the field. For instance, a Feature Engineer working in the Data & Analytics department explained that regular spreadsheets are preferred over existing solutions like MLFlow for keeping track of experiment results. The reason behind this is simplicity. Additionally, multiple other interviewees described the need to self-develop (or highly-customize) dashboards for monitoring deployed models which results in many non-reusable solutions across the company for the same problem. The authors conclude that there is a research gap between the ever-improving SOTA techniques and the challenges of developing real-world ML systems. In short, additional tool support is needed for facilitating the ML lifecycle.

In a case study at Microsoft, Amershi et al. [7] interviewed 14 people and surveyed another 551 AI and ML professionals from the company. One of the main concerns surfaced was relating to automation which is a vital cross-cutting concern, especially for testing. At the same time, a human-in-the-loop is still favoured. The survey data pointed out the difficulty posed by integrating AI, especially in the case of less experienced respondents. This was elaborated on by describing the preferences of software engineers as striving for elegant, abstract, modular, and simple systems; in contrast, data tends to be of large volume, context-specific and heterogeneous. Reconciling these inherent differences requires significant effort, nevertheless, Microsoft manages to overcome this with highly sophisticated internal infrastructure.

Using AI is not unique to large corporations, in a study conducted with the collaboration of three startups [8], the aim was to fill in the gap of understanding how professionals develop ML systems in small companies. Overall, the results showed they have similar priorities to that of large companies, including an emphasis on the online monitoring of deployed models. However, less structure is present in the development lifecycle, as one interviewee had explained: some steps are left out from time to time because they are forgotten about.

Similarly, Thiée [21] described the slow but ever-growing rate of ML adoption by small and medium-sized enterprises (SMEs). With the caveat that many more of these companies would wish to adopt data-driven approaches but are facing new challenges stemming from the domain's complexity.

[3]The terms AI and ML are often not differentiated and are used as synonyms in practice. For instance, see this study by the FDA [20]. ML is a well-defined subdomain of AI, however, most modern AI applications are also ML applications, hence, conflating the two terms may be slightly imprecise but usually not wrong.

Serban et al. [4, 22] described the results of their global surveys aiming to ascertain the SOTA in how teams develop, deploy, and maintain ML systems. In [4], they compiled a set of 29 actionable best practices. These were analysed and validated with a survey of 313 participants to discover the adoption rate and relative importance of each best practice. For example, they determined the most important best practice to be *logging production prediction traces*, however, the adoption was measured to be below 40%. In more than three quarters of the cases, newcomers to AI reported that they *partially* or *not at all* follow best practices. This tendency decreases with more years of experience, reaching a maximum adoption rate of just above 60%. In a similar fashion, Serban et al. in [22], identified another 14 best practices that concern trustworthy AI mainly through data governance. They strived to complement high-level checklists with actionable best practices. Analysing 42 survey responses revealed a familiar pattern. Most best practices have less than 50% adoption.

John et al. [23] compared and contrasted recent scientific and grey literature of AI deployments from which they extracted concrete challenges and practices. They also observed that most companies are placing a lot more models into production compared with previous years. Additionally, they pointed out that many deployment techniques are absent in contemporary literature which is speculated to be caused by the immaturity of deployment processes employed in academia. Because for instance, most models in scientific literature experience only initial deployment and are not constantly replaced or refreshed as their performance degrades over time.

Finally, in a follow-up study to [23], Bosch et al. [2] organised and structured the problem space of AI engineering research based on their 16 primary case studies. The authors noted the increasing and broad adoption of ML in the industry, while also emphasising that the *transition from prototype to production-quality deployment* proves to be challenging for many companies. Solid software engineering expertise is required to create additional facilities for the application such as data pipelines, monitoring, and logging. They defined *deployment & compliance* to be one of the four main categories of problems and described it as highly underestimated and the source of ample struggle.

## 2.3 Existing solutions

From the previous section, it is noticeable that given enough resources and at the scale of 4195 AI professionals, Microsoft managed to create a comprehensive in-house solution. A similar impression is given by Uber [24]; they built a highly sophisticated infrastructure using techniques from distributed and high-performance computing. Though, the authors note that even this solution has its shortcomings in the form rigidity (number of supported libraries and model types) but it still allows the easy extension of the system.

Given the nature of problems faced and amount of available resources, it is not surprising the both of these high-tech, Fortune 500 companies needed to, and did overcome the problems presented by deploying AI. We can learn from their approaches, nonetheless, using them may be infeasible for individuals and SMEs, thus, the issues remain for the majority of practitioners. Luckily, the open-source scene of AI/ML/DS tools, libraries, frameworks[4], and platforms is thriving. Additionally, there is a considerable number of closed-source — usually platforms-as-a-service (PaaS) — solutions next to them. Let us look at some prominent examples.

IBM's AutoAI [25] promises to provide automation for the entire machine learning lifecycle, including deployment. It is a closed-sourced, paid service which — from their documentation — seems to focus

[4]The terms *framework* and *library* will be used interchangeably in this work stemming from their vague and often holistic differentiation.

mostly on non-technical users by providing them with a UI for authoring models. The restrictions caused by the encapsulation of the entire process can be severe. The challenges of integration were emphasised above [9]. Additionally, an engineer working on Microsoft's comparable solution, the Azure ML Studio, highlighted that once users gain enough understanding of ML, such visual tools can get in their way, and they may need to seek out other solutions [7]. Unfortunately, the main value proposition of Azure ML Studio is also to provide a UI for laypeople, and it has been also set to be retired by 2024. Its successor is Azure Machine Learning which shares many similarities with AWS's SageMaker suite [26].

SageMaker offers the most comprehensive suite of tools and services; most importantly it has a set of features called *AWS SageMaker MLOps*. This provides easy and/or default implementations for multiple industry best practices described in [4, 22, 27]. Among others, it promotes the use of CI/CD, model monitoring, tracing, model versioning, storing both data and models on shared infrastructure, numerous collaboration tools, etc. Nonetheless, SageMaker does not enjoy universal adoption as indicated by the survey data. The cause of this may be the lack of self-hosting option and its relatively high prices: many companies prefer on-premise hosting for privacy and financial reasons [2]. Additionally, vendor lock-in, and possibly — in the case where it is not already used for the project — the initial effort required for setting up AWS integration could be likely deterrents.

When it comes to open-source libraries, we can find the MLOps libraries of both TensorFlow and PyTorch: TensorFlow Extended (TFX) [28] and TorchX[5]. TFX comes with a more mature set of features with the caveat that initial time-investment is needed for their setup. The features of TorchX only concern the distributed deployment to a wide range of providers, including Kubernetes (K8s), AWS Batch, or Ray [29]. There is no augmentation for the SE4ML best practices. Given the tight coupling between these libraries and their corresponding ML frameworks, they cannot generalise to models[6] or algorithms of other frameworks and technologies.

Open-source platforms also exist such as MLflow and Seldon Core. They both rely on Kubernetes to provide their features. MLflow puts more emphasis on the training phase (in deployment, it lacks a feedback loop which is essential for reaching many of the best practices), while Seldon Core focuses on the deployment stage. The latter comes integrated with a powerful explanation engine, Alibi Explain [30]. It also boasts the most comprehensive suite of features including outlier detection, online model selection (with multi-armed bandit theory), and distributed tracing. In short, it seems to be the ideal candidate for the title of *framework for robust end-to-end AI deployments*. Its only downside is the amount of complexity propagated to its clients: it is built on top of Kubernetes, and relies on Helm, Ambasador/Istio, Prometheus, and Jaeger for its features. Hence, the first step in using it is setting up a K8s cluster with all the required components, then when it comes to model deployment, a Kubernetes configuration file has to be created to make use of Seldon's Custom Resource Definition. These are smaller obstacles if the project is already built on top of K8s; however, even then, software engineers with strong cloud and DevOps background are actively required for using Seldon Core.

Additionally, increasing attention is given to ML deployments in embedded systems both from a theoretical [27] and practical [31] point of view. Prado et al. [31] survey the available deployment frameworks and end-to-end solutions including those for embedded devices. They note their inefficiencies that come from the lack of features and too much rigidity. They introduce their framework for embedded AI deployments which can be used out-of-the box but also lets users easily replace and extend its pipeline with steps to fit their changing needs and advancements of the field. While Meenu et al. [27] present and

[5]pytorch.org/torchx/latest

[6]The Open Neural Network Exchange (onnx.ai) format could be an option for overcoming these incompatibilities, however, a more universal support is needed for seamless integration.

Table 2.1: High-level comparison of popular AI deployment platforms and libraries.

| | AutoAI | Azure ML | SageMaker | TFX | TorchX | MLflow | Seldon Core |
|---|---|---|---|---|---|---|---|
| Open-source[1] | | | | ✓ | ✓ | ✓ | ✓ |
| Self-hosted[1] | | | | ✓ | ✓ | ✓ | ✓ |
| Vendor-agnostic[2] | | | | ✓ | ✓ | ✓ | ✓ |
| AI-agnostic[2] | | ✓ | ✓ | | | ✓ | ✓ |
| E2E feedback[3] | | ✓ | ✓ | | | | ✓ |
| Distributed monitoring[3] | | ✓ | ✓ | ✓ | ✓ | ✓* | ✓ |
| Online model selection[3] | ✓* | ✓ | ✓ | | | | ✓ |
| Versioning[3] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Quick setup[4] | ✓ | ✓ | | | | | |
| No DevOps dependencies[4] | | | | | ✓ | | |

[1] For privacy and accountability reasons. [2]

[2] Minimising required glue code. [9]

[3] Implementing best practices. [4, 22, 23]

[4] Easy integration into existing processes. [6, 21]

* Only partial support.

compare different architectural choices for large-scale deployments in edge-computing. They also note that: *"...there is a need to consider and adapt well established SE practices which have been ignored or had a very narrow focus in ML literature"*.

In summary, the problems expressed in Section 2.2 can be understood when looking at the available solutions. Table 2.1 shows a high-level comparison of frameworks along the dimensions in which practitioners reportedly face difficulties in the *Deployment* stage of the CRISP-DM model [32].

## 2.4 Summary

The surveys and case studies have shown the industry's continuous struggle to evolve prototypes into robust and responsible production-ready deployments. Simultaneously, platforms aiming to help overcome this challenge already exist but lack widespread adoption. The frequently recurring explanations for not adopting existing solutions surfaced in Section 2.2 revolve around their complexity and rigidity. These complaints are validated when looking at the available frameworks in Section 2.3. While using AI has become more accessible than ever, deploying remains challenging owing to the lack of any *easy-to-adopt framework for robust end-to-end AI deployments.*

The coexistence of multiple major obstacles along with their promised solutions and the lack of their wide-spread adoption leads us to believe that current frameworks are inadequate for many contexts. Thus, the answer to **RQ1** is that the complexity of deploying AI can severely hinder industrial applications even in the presence of existing frameworks. There is an unmet need for accessible AI deployment methods. The revolution brought by FLAIR, HuggingFace, and similar libraries for the domain of ML remains unmatched in the domain of AI Engineering.

# Chapter 3

# Methods

The chosen methodology for this study is Design Science which emphasises the need to design and investigate artifacts in their context [10]. It consists of a design and an empirical cycle. The purpose of the former is to improve a problem context with a new or redesigned artifact, while in the latter, the problem is investigated and its potential treatment is validated concurrently. This procedure seems fitting for our problem in consequence of its practical nature.

The design cycle shares similarities with Action Research [33] in which researchers attempt to solve a real-world problem while simultaneously studying the experience of solving said problem. As for the empirical cycle, the pragmatist approach is taken since the value of this research lies in its utility. Moreover, pragmatism adopts an engineering approach to research [34] which happens to be in line with the philosophy of design science. Additionally, as no research method is without flaws, it is imperative to try to compensate their weaknesses by applying multiple methods. Hence, the study also relies on interviews with professionals for validating the design decisions and determining the generalisability of *GreatAI*.

## 3.1 Design & empirical cycles

The aim of *GreatAI* can be summarised using the terminology of design science in the following way: *Facilitate the easy adoption of AI deployment best practices by finding a less complex framework design which is easier to adopt in order to decrease the negative externality of misused AI.*

The problem context is the difficulty in responsibly transitioning (while following best practices) from prototype industrial AI applications to production-ready deployments. With the possible treatment being libraries with high-level APIs and a set of default settings. It is important to note that *GreatAI* is merely a proof-of-concept, and its aim is to serve as a proxy for the design decisions behind it. Through this, the design can be indirectly evaluated. Hopefully, a by-product will be a library that can be effectively applied to this problem context.

The practical cases used for the evaluation are further elaborated in Chapter 5. In short, they focus on individual components of a growing commercial platform with the aim of finding tech-transfer opportunities in academic publications. The main input of the system as a whole are PDFs while the output is a list of metrics describing various aspects of each paper, such as interesting sentences, scientific domains, and the scientific contribution. The output also includes a predicted score used for ranking. This ranking is subsequently processed by the business developers of Technology Transfer Offices (TTOs) of multiple Dutch and German universities who later give feedback on the results.

Overall, this problem context carries the properties of typical industry use-cases: it utilises a wide-range of natural language processing methods, contains complex interactions between the services, benefits from the integration of end-to-end feedback, and has to provide the clients with a platform that they can rely on in their organisation's core processes. Since the final ranking affects real people, explainability and robustness are also central questions.

Before generalising, the design of the framework is iteratively refined using the feedback acquired from applying it in practical contexts which in this case is the research and development of a smaller and a more complex AI component using the work-in-progress framework. The treatment is finding a simple, less cognitively straining to use, design which still leads to high-quality deployments as defined in Section 4.2.

## 3.2 Applicability & generalisability

In order to conclusively answer **RQ3** and **RQ4**, interviews are conducted from a population of software engineers and data scientists with varying levels of professional background. Since me and my colleagues are likely to have a bias for (or against) the proposed design, the first step of checking its applicability in other practical contexts is to ask the opinion of non-affiliated practitioners.

First, before their interview, interviewees are requested to complete a questionnaire (shown in Appendix A) about their last completed AI project; the questions refer to the best practices implemented by *GreatAI* as described in Tables 6.1 and 6.2. They are also advised to take a quick look at the tutorial page of the documentation. The interviews are divided into two halves. In the first part, after a brief introduction, participants are asked to solve a real-world task by finishing a partially completed example application using *GreatAI*, they are also encouraged to think out loud so that their feedback can be noted. Successfully completing the task creates a system implementing a known number of best practices. This way, the added value — in terms of larger number of implemented best practices — can be quantitatively analysed by comparing the qualities of the finished implementation with the previously given answers.

Notes are taken throughout the interviews and subsequently extended using reflective journaling [35] combined with thematic coding. After which, the insights from the interviewed professionals are distilled using the techniques of thematic analysis [36] following the methodologies of [37] and [6]. These insights can then be combined with the numerical results to explain and elaborate on them.

The second half consists of a short survey allowing to create the Technology Acceptance Model (TAM) [38] of the problem context. The ultimate goal of the presented library is to help increase the adoption rate of best practices. In order to reach that goal, first, the library itself has to gain adoption. TAM and its numerous variations provide means of measuring users' willingness of adopting new technologies. TAM has been widely applied in literature [39] and due to its general psychological origins, it proves to be effective in other areas of technology, not just software [40].

The parsimonious version of TAM will be employed which was measured to have similar predictive power to that of the original TAM while having fewer variables [41]. Parsimonious TAM observes three interconnected human aspects that influence the actual behaviour (adoption): perceived usefulness, perceived ease of use, and intention to use. Participants are asked 10 questions corresponding to these aspects about their experience using *GreatAI*. The questionnaire is shown in Appendix B. The internal consistency of the answers is calculated using Chronbach's Alpha [42] after which the responses are reflected upon.

# Chapter 4

# Designing the framework

Providing users with a high-level of abstraction is not unheard of in the domain of practical AI/ML platforms. Many software-as-a-service products offer features for hiding the technicalities of machine learning. However — as we have discussed it in Section 2.3 — these tend to abstract away the details of both data science and AI engineering, overall hindering the development process. The design proposed here aims to tackle and simplify only the deployment related concepts.

## 4.1 Scope

As highlighted by several case studies in Chapter 2, the transition from prototypes to production-ready systems is often named as the source of unexpected struggle. Maybe it is not a coincidence that a significant portion of the SE4ML best practices should be implemented in this phase. Unfortunately, it is easy to gloss over them while tackling the underestimated difficulties of this *transition*. Therefore, the aim of *GreatAI* is to ease this step of the lifecycle, consequently, its scope is limited to the *transition* step.

There have been attempts that at least partially address this issue, however, as we have seen in Chapter 2, these have limitations either from the perspective of best practices, or stemming from their difficulty to be adopted. To have the best chance of providing an easy-to-adopt solution, the scope has to be well-defined and limited. For understanding the API of a library, users first need to understand its aim, surface, and have to become familiar with the problems it solves. Thus, focusing only on the *transition* step seems reasonable. This step is highlighted in Figure 4.1.

It is interesting to mention that there is a proliferation of platform/software as a service (PaaS/SaaS) products for deploying AI[1]. At first, these may look intriguing, however, they tend to only focus on getting code easily deployed in the cloud: AI best practices are not prioritised in this setup. Nevertheless, in many cases, it may be a suitable option to use such a service and these can also complement *GreatAI* as illustrated in Figure 4.1: first, the prototype is transformed into a GREAT service and materialised as a common software artifact implementing the best practices. Then, it is either deployed using a deployment SaaS, or by using the organisation's existing software deployment setup.

[1]Such as MLEM, Streamlit or any AutoML SaaS platform, for example, Akkio as these often have a one-click deployment feature as well.

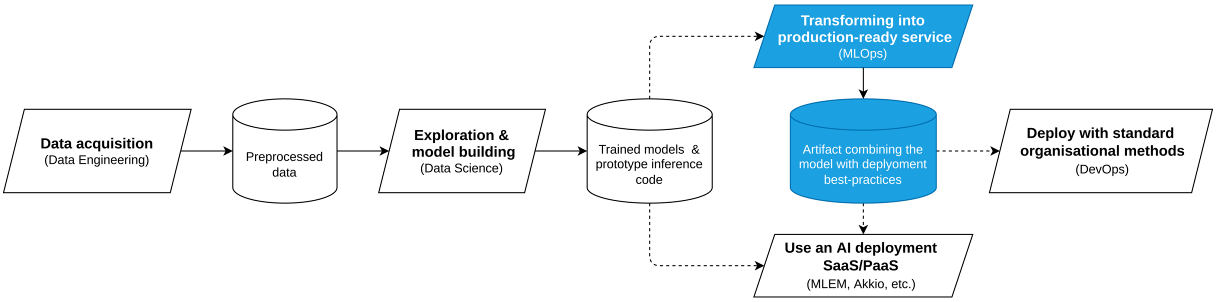



Figure 4.1: Usual process steps (based on [23]) in the development lifecycle of a data-heavy software solution. The dashed arrows denote optional paths: after a prototype has been completed, there are multiple options for its deployment. The steps with blue background show the primary scope of *GreatAI*.

## 4.2 Requirements

The best practices (which are referenced throughout the thesis) with which the design is concerned are a subset of those compiled by Serban et al. [4, 22] and John et al. [23]. The core requirements — set of covered best practices — for a software solution that has the potential of improving our problem context are presented in the following along with some explanation and clarification for each of them.

**General** Albeit not explicitly in the list of best practices, compatibility is vital in encouraging adoption. Large projects oftentimes end up depending on numerous packages, each of which may impose some restrictions on the code: since these all have to be satisfied simultaneously, this can result in severe constraints.

The open-source scene of data-related libraries is vibrant. To take the example of data validation, there are at least 4 popular choices which offer varying but similar features: Alibi detect, Facets, Great Expectations, and Data Linter [43]. The responsibility of choosing the most fitting solution falls on the user, thus, they should not be limited in this by *GreatAI*.

The programming language (PL) of the library should be its only non-general property. Fortunately, the de facto PL for data science is Python, hence, implementing the library in it should not significantly limit its applicability.

**Robustness** in software development can be achieved by preparing the application to gracefully handle errors, even unexpected ones [44]. Errors can and will happen in practice: storing and investigating what has led to them is required to prevent future ones. In the case of ML, errors might not be as obvious to detect as in more traditional applications (see the above-mentioned data validators). Even if a single feature's value falls outside the expected distribution, unexpected results can happen. In cases where this might lead to real-world repercussions, extra care has to be taken to construct as many safe-guards as feasible. *GreatAI* should support its clients in this.

**End-to-end** In this case, it refers to end-to-end feedback. That is, feedback should be gathered on the real-world performance of the system, and this should be taken into account when designing/training the next iteration of the model. Static datasets may fail to capture the changing nature of real-life and can become outdated if they are not revised continuously. A well packaged deployment should make it trivial to integrate new training data.

**Automated** The available time of data scientists and software engineers is limited and expensive. For this reason, humans should only be involved when their involvement is necessary. Steps in the development process that can be automated without negative consequences must be automated in order to achieve efficient development processes and let the experts focus on the issues that require their attention the most.

**Trustworthy** As detailed by the *Ethics guidelines for trustworthy AI*[2], human oversight, transparency, and accountability are some of the key requirements for trustworthy AI applications. For increasing public acceptance and trust while minimising negative societal impact, trustworthiness is essential.

These requirements were chosen stemming from their general importance and potential to be mostly handled (implemented) by a software framework. That is why these provide an ideal initial direction for tackling the issue. Of course, these do not cover all best practices, for instance, the ones relating to organisational processes fall outside the realm of computer science.

## 4.3 Design principles

Before diving into the concrete issues solved, let us detail the principles that should be used for implementing them in the scope of this framework. As implied in Section 4.1, the Unix philosophy [45, 46] of software design is followed. Most notably, the design goal that encourages to *write programs that do one thing and do it well.*[3]. Apart from providing a clear and simple picture of the intended use-cases for the library, this is also in line with the main notion of *A Philosophy of Software Design* [47]: APIs should be narrow and deep. A narrow width refers to having a small exposed surface area, i.e. having a small number of functions and classes in the public API. While depth implies each of them accomplishing an involved, complex goal.

In a way, the width of an API is the price users have to pay (the effort required for learning it) to use it, while the depth is analogous to the return they get from it. Having to learn little and being provided by a lot of functionality maximises return on investment (ROI), hence, developer experience (DX). The theoretical frameworks presented in *The Programmer's Brain* [48] provides us with explanations and vocabulary from psychology for arguing about the cognitive aspects of API design. In the following, two of them will be used for detailing the design principles: cognitive dimensions of code bases (CDCB) which is an extension of the cognitive dimensions of notation (CDN) framework [49], and linguistic antipatterns [50]. The former comes with a set of dimensions which describe different (often competing) cognitive aspects of code that influence one's ability to perform certain tasks on it.

Linguistic antipatterns provide guidelines for improving consistency and decreasing the false sense of consistency when there is none. Also, choosing the right names for identifiers can help activate information stored in the long-term memory which makes it quicker to comprehend and easier to reason about the code [51]. Finding the most accurate and useful names is harder than it first seems. Accuracy and usefulness are already often competing goals. The more precise the name, the longer and therefore less convenient to use [52]. In short, good names are key to good APIs; consciously considering the implications of names should be an integral part of the design process.

Nonetheless, simple APIs come at a high technical cost. The library has to implement these in a way that still allows for high performance in production [53] and avoids being tied to specific libraries or

[2]digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[3]Of course, *write programs to work together* is also very much applicable, since allowing interoperability is one of the core requirements for *GreatAI*.

technologies. Inspiration for the latter may be gained from the ML pipelines of Prado et al. [31]: they show that more freedom can be achieved with plug-and-play steps and preconfigured defaults.

### 4.3.1 Default configuration

Existing frameworks oftentimes suffer from the entanglement of numerous levels of abstractions. Instead of exposing each implementation detail and encouraging users to interact with most of them, many of these could be abstracted away in a more high-level layer. Even where configuration may be helpful for advanced users, default values can still be chosen automatically while providing an override option where necessary.

For example, tracing the evaluations and the model versions used in a distributed fashion is very much expected of a trustworthy system. Hence, turning this feature on by default but allowing opting-out from it can result in less scaffolding required from the library's users. It also decreases their up-front cognitive load which by definition flattens the learning-curve [48]. Similar features can be imagined for providing a service API for the algorithms and for giving feedback, marking outliers, etc.

Being *automated* is listed as a requirement but it is imperative to only automate for simplifying and not for hiding decisions. More precisely, guessing must not be a part of automation. For instance — an otherwise incredibly useful WebGL library — TWGL.js has a feature for automatically guessing the type of vectors based on their names. If it matches the `/colou?r/i` pattern, it is treated as a vector with three components[4]. It is easy to imagine that this can help in certain scenarios, but it does so at the cost of immense confusion when correctly renaming a variable breaks the application. In CDCB, this equates to scoring high on the dimension of *Hidden dependencies* and low on *Visibility*.

Learning from this, any kind of guessing must be avoided for creating a pleasant API. However, this conflicts with providing defaults for each configuration value. Even if these would be reasonable defaults derived from educated guesses, they are still merely guesses. Nevertheless, if the users were required to specify each configuration option, that would lead to considerably more boilerplate code. This verbosity is captured by the *Diffuseness* dimension of CDCB and, of course, should be minimised.

To resolve this conflict, *GreatAI* should have recommended values instead of defaults. This can mean a context object (as suggested in [47]), which contains the result of each design consideration that has to be made for a service's deployment. If not configured manually, the recommended values are applied automatically, just like defaults. The values chosen for each parameter must be clearly highlighted. Coming from the library's single responsibility, the number of parameters should not be immense, hence, the user can be expected to comprehend them instead of just being overwhelmed and skimming it.

This way, the library attempts to notify its user about the existence of these decisions but does not force them to manually decide. As a result, no initial configuration is needed for starting out with the library (high *Provisionality*, low *Diffuseness*) and the dependencies are not hidden since they are explicitly highlighted.

### 4.3.2 Documentation

Little value can be derived from software without good documentation; undoubtedly, good documentation is a prerequisite for adoption. Documentation comes in many shapes: modern integrated development environments (IDEs) tend to show a popup of a function's description when requested (on mouse hover

[4] github.com/greggman/twgl.js/blob/e3a8d0ed09f7f5cd4be0e4cb5976081c2b5013aa/src/attributes.js#L139

for instance), at the same time, a more comprehensive online manual and example projects are also still expected. But descriptive error messages can be also viewed as documentation.

The library must have quality documentation for all categories. Accordingly, for structuring it, the *Diátaxis* philosophy is preferred [54] which prescribes dividing documentation into 4 parts along 2 axes: practical-theoretical and passive-active consumption. The four quadrants derived from this are: tutorials, how-to guides, reference, and explanation.

Once again, we might notice two competing interests: the level-of-detail and the length of the documentation. For example, FastAPI[5], a popular Python web framework, has extensive descriptions and explanations on all topics related to Python's import system, the HTTP protocol, concurrency, deployment, etc. The actual framework's documentation is sprinkled over these very broad topics. This is certainly helpful for beginners to acquire knowledge from a single place. Nevertheless, this high-level of accessibility actually hinders the process of finding the relevant sections (in CDCB, this shows a trade-off between the support of *Searching* and *Comprehension* tasks). Diátaxis' take is that linking to external resources about the library's domain are welcome, but the documentation must have a single responsibility: describing the library itself.

A large portion of software documentations is automatically generated from source code. This has the advantage of always keeping it in sync with code changes, however, it might also signal that the API is too large because it is inconvenient for the developers to document it by hand. Striking the right balance between handcrafted and automatically extracted documentation may be a vital component of good documentation.

When it comes to example code, showing at least a minimal starter code and the way of customising it has to be showcased front and centre. It is a well-known observation that developers only read documentation when they are stuck and there might be some merit to this. Making them not get stuck — by providing starter code from which they can explore the API using IntelliSense-like solutions — should be preferred. For example, another widely popular Python web framework, Flask[6], at this time, has 324 homogeneously styled links on its landing page. Out of these, only 2 lead to the quick-start code. Of course, it is not hidden, but I argue that the DX could be improved by displaying where to start more prominently.

### 4.3.3 Developer experience

Subjectively, a key component of good DX is *Progressive evaluation* through which development can become a highly iterative, experimental process. This is well understood by popular data science tools, such as Jupyter Notebooks. *GreatAI* also has to support some level of this, for example, in the form of auto-reload on code changes. Further key ingredients for good DX are consistency and discoverability. To give one more example, the MySQL connector's Python implementation[7] has a cursor object which exposes a `fetchone` method. Even though this naming scheme is not conventional in Python since it does not follow PEP 8, at least the API is intuitive: changing `sql_cursor.fetchone()` to `sql_cursor.fetchall()` returns all items instead of just one. Using good and consistent names is the key to good DX.

At the same time, Python codebases are rarely strictly object-oriented (OO), they are a mix of the functional, data-driven, and OO paradigms. Consequently, relying on classes for grouping related functions

[5] fastapi.tiangolo.com

[6] flask.palletsprojects.com/en/2.1.x

[7] dev.mysql.com/doc/connector-python/en/

is not always desirable. Therefore, it is even more imperative to name similar functions similarly. This helps discoverability and chunking [48] which amount to quicker comprehension.

There is one more reason to prefer consistency: humans have a limited short-term memory (STM) [55]. Even though flags as function parameters are frowned upon by some [56], they are useful, especially, when configuring libraries. However, if there is no convention for the default value of a flag, clients have to remember the flag's name and initial value at the same time, quickly overloading their STM. Thus, in the codebase, all defaults must be the same, let us say, `False`. Sometimes, it can result in a *disable* prefix which may turn into a double negation, nevertheless, users should not ever encounter this themselves since the doubly-negated version is the default, thus when overriding it, it is only singly-negated. This approach also implies, something may be recommended to be turned on by default.

# Chapter 5

# The ScoutinScience platform

The core product of ScoutinScience B.V. is its platform[1]. The clients are technology-transfer offices of Dutch and German universities, government organisations (e.g.: Wetsus), and corporates (e.g.: Heraeus Group, Ruma Rubber B.V.) who wish to extend the scope of their R&D activities. ScoutinScience connects to multiple data sources of academic publications and integrates them into a single database. Each new publication is evaluated with a suite of AI components that ultimately determine its technology transfer potential. Other features are also extracted that help the users get a quick overview of the authors, topics, and contributions of a given piece of research.

Each client organisation gets to see a different filtered view of this database ranked by the predicted probability of technology transfer opportunities being present. The main motivation is to make these business developers' and other professionals work more efficient by showing them which papers have the highest chance of being considered interesting by them.

To achieve this, we have a service-based architecture [53] on the backend-side — apart from the data integration, communication, and business logic — it is made up of services wrapping simpler (phrase-matching, Naïve Bayes) and more sophisticated (conditional random fields, transformer) models. As we will soon see, these can also depend on each other, for instance, based on the predicted scientific domain, a different model can be chosen for scoring certain aspects of papers.

I was among the first engineers on the team which has grown considerably in the past two years. While architecting, designing, and integrating more and better models into our software solution, I experienced the same difficulties as were described in Chapter 2. The gap between prototypes and production-ready services is larger than it seems. It is also larger than it should be. This had motivated me to investigate the state-of-the-art and I have found that it is insufficient in many cases. Since the ScoutinScience platform is a quite typical example of applying AI in the industry, it will serve as the real-life case, problem context, and testbed for attempting to design a solution which can hopefully advance the state-of-the-art.

In this chapter, the process of designing *GreatAI* is described along with how it fits into real-life use-cases. First, a simple experiment is presented which leads to the implementation of a software service, subsequently, as the feature-set of the library grows and matures, a more complex component is developed. Lastly, the final version of the design is presented and qualitatively evaluated to verify how well it satisfies the requirements described in Section 4.2.

[1] dashboard.scoutinscience.com

## 5.1 Domain classification with Naïve Bayes

Using different models for slight variations of the same problem is quite commonplace in the industry. For instance, UberEats has a vast, hierarchical set of models for every country, region, and city for calculating the estimated time of delivery [24]. We have also found that in order to best process an academic publication, knowing its domain is essential. The reason for this can be (among others) the wildly different vocabularies of different domains. For example, the term *framework* in computer science almost always refers to a software artifact (usually implying high tech-transfer potential), while in most other domains, *framework* is used to describe theoretical models that are less central to practical applications. Of course, it is not merely the meaning of the terms but more importantly: their distribution that varies significantly. Therefore, the topic of this section is to design and develop a domain prediction model for academic papers.

### 5.1.1 Background

Fortunately, this is one of the oldest text classification tasks. In fact, Maron introduced the Naïve Bayes classifier in 1961 [57] for exactly this purpose: classifying documents' subjects. However, it is still an active problem when it comes to academic texts as indicated by Elsevier funded research carried out by Rivest et al. [58]. They created a 176-class classification problem for comparing bibliometric and deep-learning approaches but this comparison is made difficult because 44% of the labels are *assigned suboptimally* in the ground-truth dataset.

Prior work evaluated SciBERT [59] — a BERT [14] model pretrained on academic publications — on a simpler version of the task in which the domains of sentences[2] have to be decided[3]. It achieved an F1-score of 0.6571 after being pretrained on the Semantic Scholar Corpus (SSC) [61] and finetuned on the train split of the Microsoft Academic Graph (MAG) dataset [62][4]. To the best of my knowledge, no other published work exists on this sentence-classification task. This may be explained by the lack of practical relevance and contrived nature (uniform label distribution) of the task as we will see in the next subsection.

**Design note** After getting familiar with the context, it is time to focus on experimenting and developing our domain prediction service. At the same time, the difficulties encountered should be noted and integrated into *GreatAI*'s design.

### 5.1.2 Data

Two datasets will be considered for the experiments. SciBERT's MAG and the SSC. The former is used to compare the results with SciBERT's, while the latter is utilised for training a model for production purposes because it has 19 labels compared with MAG's 7 and it also contains abstracts instead of just sentences, thus, it is more fitting for our practical use-case.

[2]Sentences are more appropriate units for processing due to SciBERT's maximum token length of 512 which comes from its attention mechanism's quadratic complexity [60].

[3]paperswithcode.com/sota/sentence-classification-on-paper-field

[4]SciBERT was applied to a preprocessed version (github.com/allenai/scibert/tree/master/data/text_classification/mag) of this dataset.

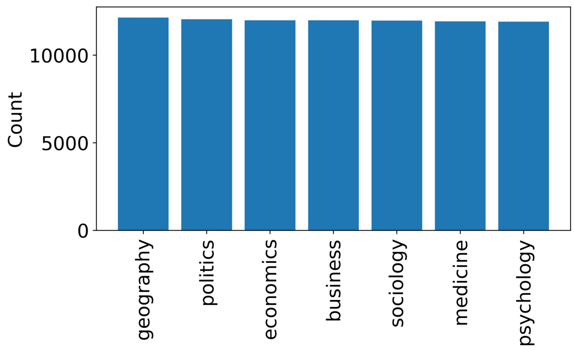



Figure 5.1: Class distribution of the MAG [62] dataset's 84000 sentences in its *train* split.

SciBERT's version of the MAG dataset has 84 thousand and 22.3 thousand sentences in its train and test splits respectively. These are mostly in English and have all punctuation and casing removed. Each sentence is classified as belonging to one of seven fields. Figure 5.1 shows that the classes have a uniform distribution.

SSC is much larger: it contains over 80 million abstracts. Having more data certainly helps in sampling the term distribution more accurately, nonetheless, the law of diminishing returns applies, especially when using simple models. Therefore, the data will be randomly downsampled to leave us with a more manageable couple of hundreds of megabytes of abstracts. We can see the distribution of class labels in Figure 5.2. The dataset is considerably less balanced: *medicine* is by far the most voluminous field.

**Where should we store this data?** "On my machine" seems like an easy answer. However, if we have a team working with the data or it has intrinsic value, it must be stored in an easy-to-access, potentially redundant way. Serban et al. [4] expressed this need in the following best practice: *Make Data Sets Available on Shared Infrastructure (private or public)*. Meanwhile, wherever data is stored, it should be also versioned to satisfy the next best practice: *Use Versioning for Data, Model, Configurations and Training Scripts*.

MAG needs no further preprocessing if we aim to match SciBERT's setup [59]. But since SSC contains heaps of metadata, the relevant parts have to be extracted and preprocessed. In this case, these are the concatenation of the abstract's text and the paper's title along with the paper's domains (there can be multiple domains for a single paper: it is a multi-label classification task). Lastly, the non-English entries are discarded because we only expect to process papers in English.

**How should we preprocess the data?** These simple processing steps (filter, map, project) are almost always present in the data science lifecycle. For example, cleaning the input text from various HTML, OCR, PDF, or LaTeX extraction artifacts is almost always necessary for text analysis. This is captured in the AI best practices collection under the following category: *Write Reusable Scripts for Data Cleaning and Merging*. Also, the best practice of *Test all Feature Extraction Code* is somewhat applicable: the applied processing steps must not introduce unwanted artifacts.

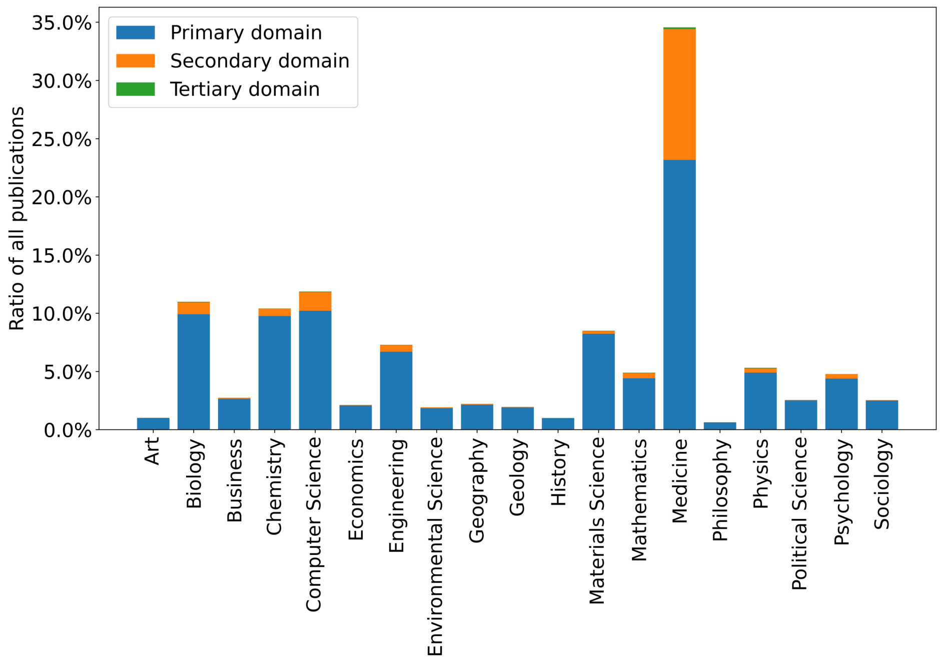



Figure 5.2: Label distribution of the Semantic Scholar dataset [61]. Each publication may be assigned at most 3 domains.

### 5.1.3 Methods

Our aims are twofold: (1) to evaluate a sentence classification model on MAG and compare it with prior art; and (2) to retrain and apply this model for classifying publication metadata (including abstracts). This would allow the ScoutinScience platform to select an appropriate processing pipeline which has been trained on a matching vocabulary (and domain) for each publication.

It seems reasonable that only considering the distribution (frequencies) of individual terms may be sufficient. To test this hypothesis, a unigram language model (Multinomial Naïve Bayes) is constructed and its accuracy is compared with SciBERT's. The former definitely aligns with the advice to *Use The Most Efficient Models.*

Using the MNB implementation of scikit-learn [17], it only took a couple of lines to create, hyperparameter-optimise, and test a text classifier. Including data loading and visualisations, it takes 71 lines of code (LOC) to be more precise. [5] This further proves relatively how simple it is to use standard packages. The code can be considered for satisfying the *Automate Hyper-Parameter Optimisation* best practice, since it also implements an automated hyperparameter sweep.

The sentences are tokenised into words and vectorised with TF-IDF (with logarithmic term frequency) [63], the hyperparameters found via 10-fold cross-validation on the *train* split lead to filtering out tokens which occur in fewer than 5 documents or more than 5% of the documents.

[5] The code is available at great-ai.scoutinscience.com/tutorial.

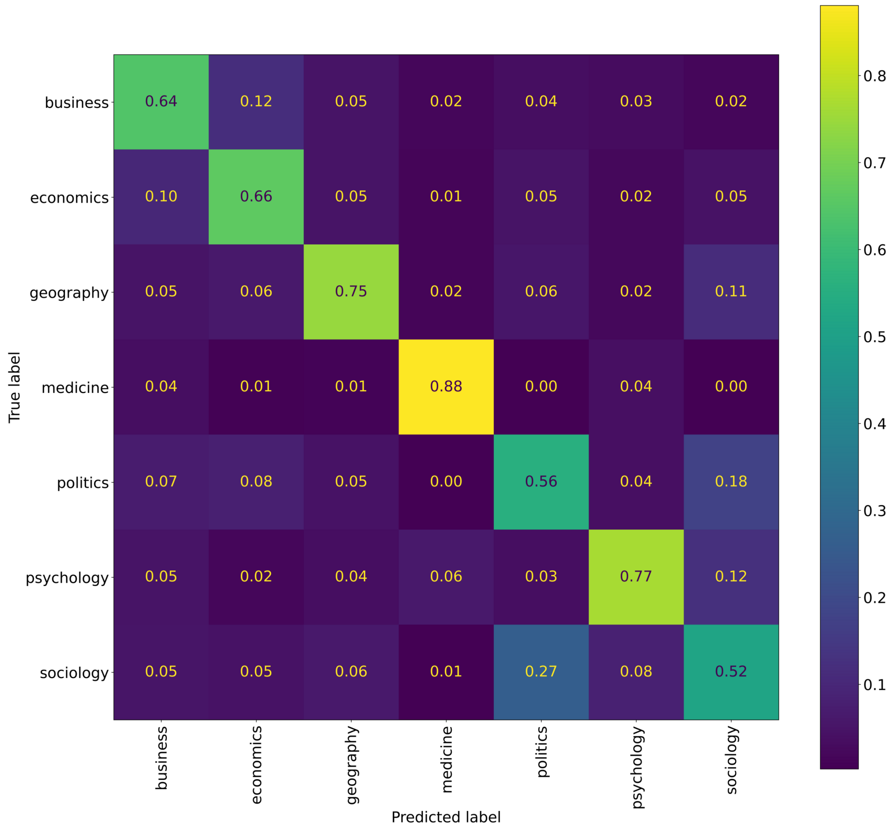



Figure 5.3: Confusion matrix of a Naïve Bayes classifier on the MAG dataset's sentences. The matrix is normalised column-wise. Notice, how most mistakes happen between semantically similar classes, for instance: *politics – sociology* or *business – economics*.

**What could be automated here?** As discussed in Section 2.1, libraries exposing algorithms and even SOTA models can already be considered mature and accessible. In this case, only scikit-learn was utilised, but subjectively, most popular libraries have a similarly easy to use API. Therefore, I see no urgent need for further action regarding the *experimentation* step of the lifecycle in connection with the AI best practices.

### 5.1.4 Results & Discussion

When this model is applied to the *test* split of MAG, we get the confusion matrix of Figure 5.3. This Naïve Bayes classifier achieves a whopping 0.6795 F1-score. This is 2.3% more than SciBERT's on the same dataset. Thus, it seems, MNB clearly outperforms SciBERT for this particular use-case: it is not only more accurate, its model is magnitudes smaller, while it is also considerably faster to train (or finetune in the case of SciBERT) and use (its running time is in the order of milliseconds per publication). It also has no upper-limit on the input length. Thus, this experiment validates the choice of picking MNB for the task over SciBERT.

It is, of course, not entirely surprising that the sophisticated transformer architecture of SciBERT is not necessary for a plain task like this. Apart from phrases, the relation between separate words of a

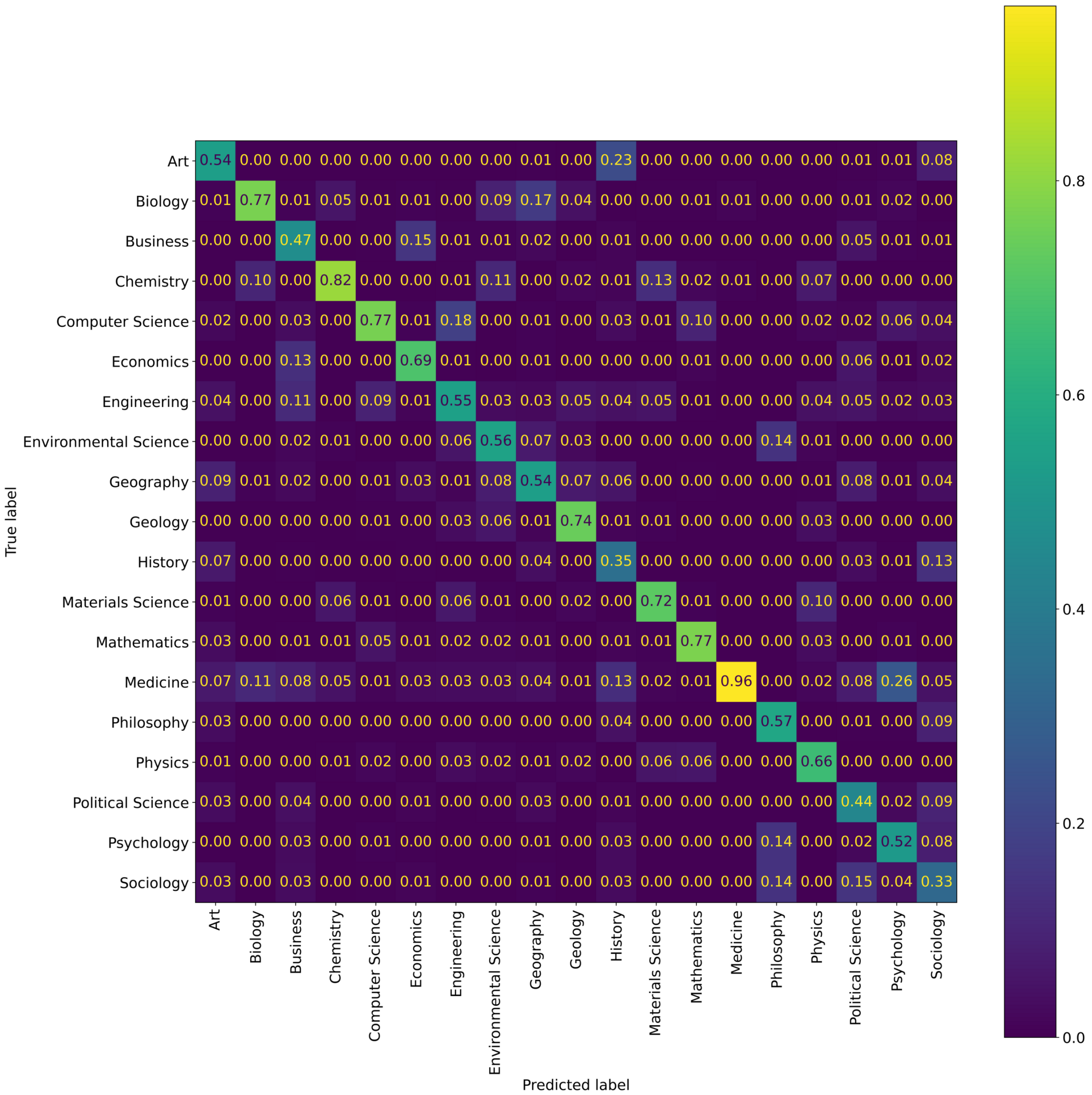



Figure 5.4: Confusion matrix of a Naïve Bayes classifier on the SSC dataset's sentences. The matrix is normalised column-wise. Notice, how most mistakes happen between semantically similar classes, for instance: *philosohpy – sociology* or *history – art*.

sentence do not carry nearly as much discriminative power as the identity of the terms [64], hence, there is little reason for using an attention mechanism. The fact that SciBERT even works in any way on this task is already a testament to its general applicability. Nevertheless, this short experiment has proved that we can safely opt for using MNB for production.

Since Multinomial Naïve Bayes is best at returning a single label and SSC has multiple labels per datapoint: for evaluation purposes, it is checked whether the returned label is contained in the labels of the ground truth. On this dataset, MNB achieves lower macro-average F1-score which is 0.59.[6] The weighted-average F1 is 0.70 and the overall accuracy is also 70%. The substantial difference between the macro and weighted averages come from the unbalanced distribution of the labels.

The lower F1-score is not surprising because there are more than twice as many classes in this dataset. Additionally, the mistakes made are defensible when we look at Figure 5.4: most of them are between close or related classes.

This is the usual point where papers conclude: a proof-of-concept/prototype has been built and its performance demonstrated, measured — and usually — explained. Nonetheless, in an industrial setting, our problem is far from being solved: it has yet to be deployed.

### 5.1.5 Deployment

First, an inference function needs to be written that can take an input on the fly and calculate a corresponding prediction. Since we aim to follow the best practices, namely: *Explain Results and Decisions to Users* and *Employ Interpretable Models When Possible*, giving an explanation of the results is expected. Fortunately, with our simple model it is easy to determine the most influential weights, thus, words; the explanations are derived by taking the top 5 tokens from the input text ranked by their feature weights. The last deployment step may be to provide access to our model for others.

**How do we provide an interface for the inference function?** We either have an offline or online inference workflow (or both). For the former, we have to provide a way to use it in batch processing; a simple Python function may be adequate for this purpose, though, allowing it to be easily (or automatically) parallelised would make its consumers' DX better. If it is an online workflow, we must have a service running continuously and accepting input at any time. This can be achieved by a remote procedure call (RPC) interface, or more commonly, a web API. Developers usually refer to these as REST APIs, sometimes, they even follow the conventions of REST. Either way, we must develop a wrapper over the service in order to make it available for other internal/external consumers.

According to the body of research on the adoption of best practices, this is where many real-world projects conclude. This also happens to be **the gap**. Believing that solely focusing on the research and experiments is good enough is a fallacy: when following this approach, the deployment step ends up being a rushed attempt of wrapping the *AI* and putting it in the production environment. This is inarguably a deployment. However, it likely follows very few of the best practices which can lead to suboptimal real-life performance, lack of accountability, lack of opportunity to improve, and possibly an overall negative societal impact.

[6]The code for this is available at great-ai.scoutinscience.com/examples/simple/deploy.

**How could we implement more best practices?** The most notable missing software/operations features are the lack of automated deployment, automated regression testing, online monitoring, persisting the traces, graceful error-handling, taking feedback on the results (if possible in the use-case), calculating the online accuracy based on the feedback, and retraining the model if necessary using novel data. These all correspond to best practices.

## 5.2 Bridging the gap with GreatAI

First, let us revisit the library's scope for clarification. As concluded in Section 4.1, *GreatAI* should ease the *transition* step between prototypes and production-ready deployments. However, this leaves open the question of what constitutes to this step? There are cross-cutting concerns, for example, feature extraction is implemented and used in the training phase, but it is also deployed alongside the model. The robustness criterion has to be met by this procedure even though its implementation is only in focus in the earlier stages of the project. Since having an untested function deployed into production can have severe repercussions, I conclude, assuring its correctness lies within the scope of *GreatAI*.

This section briefly explores how the problems raised can be solved using *GreatAI*, and the API it provides in order to best fit the needs of its users. We first focus on the aspects of data, then, we discuss the utility of helper functions, and lastly, the automated wrapping of services.

### 5.2.1 Handling data

The obstacles coming from the intertwined nature of different models is widely recognised [6, 7, 9]. This can lead to non-monotonic error propagation, meaning that improvements in one part of the system might decrease the overall system quality [7]. The importance of schema versioning in an environment of rapidly changing models and transformations is highlighted for a specific use-case in [65] and more generally by the *Use Versioning for Data, Model, Configurations and Training Scripts* best practice. These emphasise the requirement for versioning models and, in general, data.

There are two kinds of data storage needs we have to address: training data and trained models. Because our code is probably already tracked under Git (and likely synchronised with GitHub), using the Git Large File Storage (LFS)[7] might seem intriguing. However, it is a paid (and surprisingly expensive) service of GitHub especially when we factor in the expected sizes of the models and training data with the fact that the only way to remove files counting towards our quota is to delete the entire repository.

An open-source tool, the Data Version Control (DVC)[8] provides a nearly perfect alternative. It comes with a command-line interface (CLI) inspired by Git's, and it can be integrated with several backend storage servers. Its only downside is, of course, that it is one more tool that increases the complexity of the project and the initial setup time. If this is an acceptable price to pay, then I personally recommend opting for DVC. Nevertheless, if this may prohibit a team[9] from properly handling data according to the best practices, I present a simpler solution.

The complexity of an API can be decreased by relying on its users preexisting knowledge and known patterns [48, 47]. Therefore, we can reuse familiar APIs, such as the `open()` method from Python.

[7] git-lfs.github.com

[8] dvc.org

[9] As was the case with MLFlow tracking in an ING team described in Section 2.2.

Therefore, a method is proposed which provides the same interface, however, the backing storage can be a mixture of local disk space, S3-compatible storage, MongoDB, or any other storage backend. It provides a superset of `open()`'s interface[10]: the same parameters can be used with it resulting in similar observed behaviour. The expected features: versioning, progress bars, caching, garbage collecting the cache, automatically deleting old remote version are all present and come with recommended — but easy to see and change — configuration.

Easing development is not merely about automating everything but also making the code easy to change (which is the *Viscosity* dimension of CDCB). Going from opening a local file on the disk with the built-in open method, to opening a file from S3 is as easy as changing `open('file.txt', 'w')` to `LargeFileS3('file.txt', 'w')`. In the case of the latter, an additional `version` keyword argument can be also given to lock ourselves in using a certain version which is very much desired in the case of models.

### 5.2.2 Utilities

It is easy to notice multiple recurring tasks when it comes to processing text. Cleaning it from various extraction artifacts and normalising characters is one of the most common. But splitting sentences, language tagging, robustly lemmatizing are also often recurring tasks. Because having reusable and tested feature extraction code covers two best practices, it seems straightforward that a utility module could be created for this which can also be extensively tested by means of unit testing.

This is exactly the motivation behind `great_ai.utilities`. Extra care has to be taken not to overfit these utilities on the cases considered in this chapter; however, I believe these are versatile enough to be helpful in many text-related contexts. A conclusive answer to this assumption will be found during the interviews.

Implementing the unit tests uncovered multiple edge cases and even runtime errors, hence, the merit of *Test all Feature Extraction Code* best practice is unequivocal. There is one more best practice that could be partially covered here, especially, because its solution also helps both during batch inference, but also at training/feature extraction time: *Enable Parallel Training Experiments.*

A function called `parallel_map()` is implemented which closely mimics the API of the built-in Python function: `map`. And it exemplifies how even a close to trivial function is able to improve the DX by magnitudes. Rooted in the global interpreter lock (GIL)[11] of CPython, in almost all cases, multi-threading does not lead to higher performance of CPU-bound tasks. For this purpose, multiprocessing has to be used. Fortunately, the built-in `multiprocessing` library has a great API, however, it still takes about a dozen lines to do a parallel mapping task with a progress bar. This can deter people (at least me) from taking advantage of more than just a single CPU core during exploratory experimentation. With `parallel_map()`, this challenge becomes a single-line, routine task.

### 5.2.3 Deployment approach

Some of the expectations one might have for data-intensive (such as AI) software are similar to that for software in general. These are also captured by the best practices: *Use Continuous Integration*, *Automate Model Deployment*, and *Enable Automatic Roll Backs for Production Model* to name a few. It is important to notice that these have been already solved by software engineering, more specifically, by the DevOps

[10] docs.python.org/3/library/functions.html#open

[11] wiki.python.org/moin/GlobalInterpreterLock

paradigm [66]. In line with the findings of John et al. [23] on the SOTA of AI deployments, I suggest we wrap the applications in a format which is more compatible with existing DevOps toolkits. Instead of reinventing the wheel, we should rely on more established DevOps best practices for implementing the SE4ML deployment best practices. Besides, organisations are expected to have their deployment processes for classical applications, thus, allowing them to reuse those for AI applications seems to be the most convenient approach.

Based on personal empirical evidence, three types of software artifacts are identified (in the context of Python) for which a wide range of established practices exist. WSGI server[12] compatible applications, executable scripts, and Docker Images[13]. To achieve this, *GreatAI* provides a compatibility layer between simple Python inference functions and all the above common artifacts. Taking functions as input for the first step also satisfies the requirement to be **General**. Nevertheless, in order to also allow customisation, additional configuration, metadata, and behavioural specification can be given as well.

The main advantage of the wrapping approach is that it does not require any input from the clients (by default). I opted for a decorator [67] which lets users wrap their function by adding a single additional line of code as shown in Listing 1. After which the created WSGI application can be accessed through the `greeter.app` property where `greeter` is the identifier of the user-defined function. A CLI script (`great-ai`), along with a `Dockerfile` are also provided to cover the other two deployment artifacts.

```python
from great_ai import GreatAI

@GreatAI.create
def greeter(name: str) -> str:
    return f"Hello {name}!"
```

Listing 1: Simplest example using *GreatAI* for wrapping a function. In practice, `greeter` probably would be the inference function of an ML model.

Coincidentally, deployment best practices can be easily implemented in this wrapper layer. In the first iteration these are: input validation, persisting traces, online monitoring, and generating documentation. Input validation may be used to *Check that Input Data is Complete, Balanced and Well Distributed*. Traces are important for both *Log Production Predictions with the Model's Version and Input Data* and *Provide Audit Trails*. However, traces can also indirectly help **Robustness**, because even production systems cannot be expected to be perfect. Saving and letting the users filter on encountered errors while allowing them to correlate it with the input causing it is imperative for facilitating debugging. Lastly, monitoring and documentation correspond with helping best practices: *Continuously Monitor the Behaviour of Deployed Models* and *Communicate, Align, and Collaborate With Others* respectively.

To allow customising the service's behaviour to fit different use-cases, the default configurations can be overridden by calling some functions of the library. An example of this can be seen in Listing 2, while more details of the semantics can be found in the documentation[14].

[12] peps.python.org/pep-3333

[13] docs.docker.com/registry/spec/manifest-v2-2

[14] great-ai.scoutinscience.com/how-to-guides/create-service

```python
from great_ai import GreatAI, parameter, use_model, log_metric

@GreatAI.create
@parameter('positive_number', validate=lambda n: n > 0)
@use_model('secret-number', version='latest')
def add_to_secret_number(positive_number: int, model: int) -> int:
    """This docstring will be exported as documentation."""
    log_metric('log directly into the Trace', positive_number * 2)
    return secret + positive_number

assert add_number(1).output == 5
```

Listing 2: A simple *GreatAI* service with behavioural customisations. In practice, the function would probably be the inference function for an ML model.

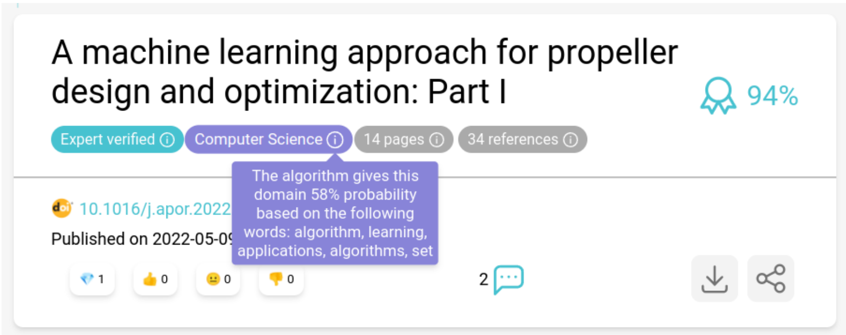



Figure 5.5: Screenshot of the domain prediction integrated into the ScoutinScience platform where it is used as a filtering option.

### 5.2.4 Summary

After implementing some features of the library, it can be already used for deploying the previously discussed domain prediction model. In this case, online prediction is expected, hence, the REST API-based deployment is chosen which is created by `GreatAI.create` and packaged in a Docker image. This image can be instantiated by the company's existing DevOps pipeline and cloud infrastructure. At the end, users can see one more tag in the header section of publication evaluations where they can also see the explanation behind the model's decision as demonstrated in Figure 5.5. Let us now explore how it fares in a more complex case.

## 5.3 Text summarisation with SciBERT

The ScoutinScience Dashboard contains a full-page evaluation view for academic publications. On this, the known metadata, historical trends about the paper's topics, social media mentions, a PDF viewer showing the document, and other augmentation tools are displayed. One of these is the *Highlights* section, which aims to summarise the paper from a technology-transfer perspective.

The current approach uses a simple heuristic based on a set of phrases selected by business developers and extended by the help of a word2vec model [68]. The user feedback deemed this implementation slightly helpful but not adequate for providing an accurate overview. Thus, this is the baseline that I attempt to improve on in this section.

Compared with Section 5.1, this time around, the toolset of *GreatAI* is available at our disposal. Hopefully, this will streamline the development and — especially — the deployment. Given its arguably higher complexity, the experiment falls closer to industrial use-cases, and hence, can give more accurate feedback on how to further improve the API.

### 5.3.1 Background

Automatic text summarisation (ATS) is also one of the earliest established tasks of text analysis and boasts numerous promising results [69]. Text summarisation is usually divided into extractive and abstractive approaches. Even though the latter can lead to more fluent summaries, it is also prone to hallucinate content that is unfaithful to the input [70]. For this reason, extractive techniques are preferred in this case.

Our problem requires generating a special type of summary: it must only concern a single aspect (tech-transfer) of the document. Aspect-based text summarisation has also seen some progress over the last decades [71, 72] but these methods require concretely defined topics. Unfortunately, *tech-transfer potential* is anything but a clear topic definition.

Numerous discussions and interviews with business developers over the last two years made it clear that there is no universally agreed on definition of it. At least, all of them agree that they know it when they see it. Additionally, most of them agree that they can confidently make a decision at the granularity of sentences. This gives rise to an obvious idea: show the experts something that they can annotate. Because the time of experts is valuable, and relevant sentences are few and far between, extra care needs to be taken to improve the ratio of positive examples in the dataset. The research of Iwatsuki Kenichi on formulaic expressions (FEs) [73, 74, 75, 76] provides a promising direction to do so.

A formulaic expression is a phrase with zero or more "slots" which when filled appropriately, leads to expressing a certain intent. In the context of scientific text, an example[15] could be: `it was not until * that`. The asterisk can be substituted with multiple terms and the intention of this expression is (likely) to describe the *History of the related topics*. Iwatsuki et al. identified a set of 39 intentions, compiled a manually labelled dataset [73], and developed multiple approaches for automatically extracting and classifying formulaic expressions in large corpora [75, 76].

### 5.3.2 Methods

In order to compile a new dataset, experts are asked to judge sentences that passed an *intention check*. This pooling approach is commonly used in the field of information retrieval [77]. The filtering is expected to sieve out sentences that are probably not relevant from a technology-transfer perspective using Iwatsuki's formulaic expression intention classes. Subsequently, relevance judgements — in the form of *interesting* or *not interesting* labels — are gathered for the remaining sentences. This method turns the extractive summarisation into a binary classification task for which a SciBERT model [59] can be fine-tuned. Ultimately, the summaries are derived from sentences that are selected by the classifier trained on the experts' annotations.

We have to note two possible shortcomings of this setup: firstly, the FE intentions are assumed to be strongly correlated with the sought-after aspect, this may or may not be true. Secondly, only the individual relevance of the sentences is considered instead of the overall relevance (utility) of the summary.

[15] Taken from the ground-truth data available at github.com/Alab-NII/FECFevalDataset.

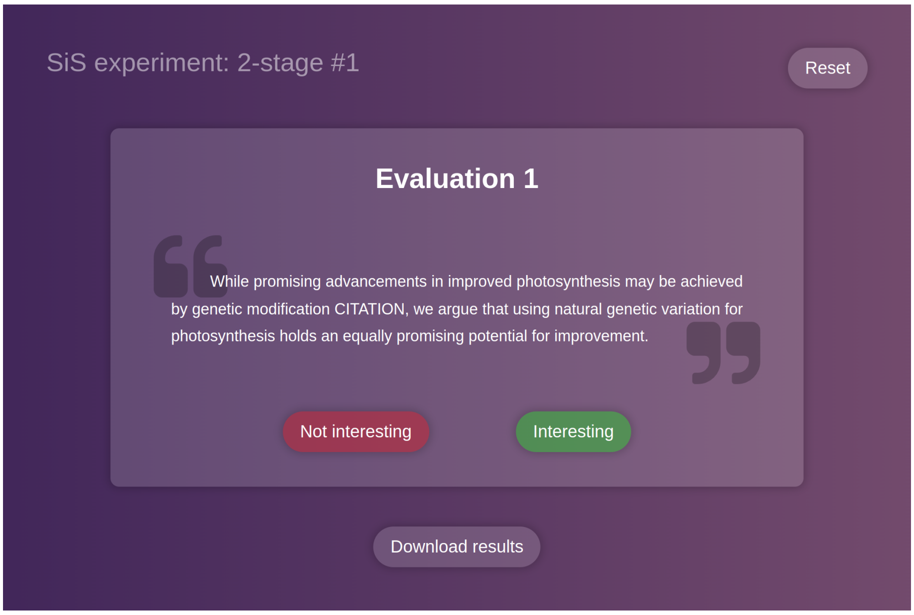



Figure 5.6: Annotator UI showing a single sentence and the two possible labels that can be assigned to it based on its relevance to technology transfer.

Nonetheless, it is expected that stemming from the length of the documents and the sparseness of the selected sentences, that any combination of them is likely to have low redundancy.

### 5.3.3 Results

For the first iteration, 1500 sentences were selected for 2 experts to annotate in a binary fashion according to strict guidelines. An example is shown in Figure 5.6. Afterwards, for measuring the interrater agreement, Cohen's kappa [78] is calculated as shown in Equation 5.1. Which turns out to be **0.4310** for the two annotators. This happens to be just above the lower end of *moderate agreement*. However, we have to note that the original quality ranges are often criticised for being too relaxed [79]. However, in the case of summarisation, Verberne et al. [80] argue that reasonable end-quality can be reached even when the interrater agreement is relatively low. The ground truth is determined by taking the logical disjunction of the annotations.

$$\kappa_{agreement} \equiv \frac{p_{observed} - p_{expected}}{1 - p_{expected}} = 1 - \frac{1 - p_{observed}}{1 - p_{expected}} \tag{5.1}$$

The next step is finetuning SciBERT with the help of HuggingFace transformers [13]. The data are divided into training and test splits with a ratio of 4:1. From the train split, a validation split is also derived which is used for early stopping. The objective function is the F1-score of the positive class and the early stopping patience is 5 epochs. The learning rate is $5 \times 10^{-5}$ and AdamW [81] is used for optimisation with a weight decay of 0.05. The code can be found in the documentation[16], it is surprisingly slightly shorter than the code of Section 5.1.

[16] great-ai.scoutinscience.com/examples/scibert/train

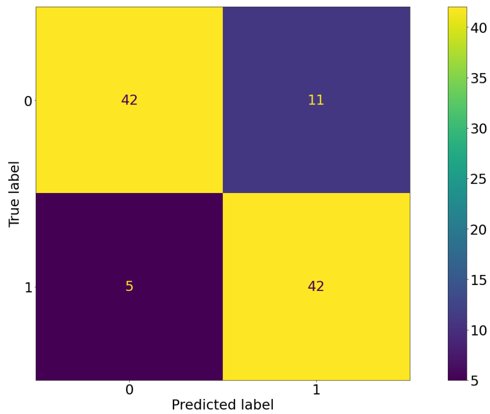



Figure 5.7: Confusion matrix of the fine-tuned SciBERT model on the *summary candidate sentences* dataset. The values are globally normalised and represent percentages.

**Reproducability** Reproducible experiments are generally preferred. It is easy to forget to set some seeds values and, for example, end up with different datapoints in the test-train splits during training and validation in a Continuous Integration (CI) pipeline. To facilitate reproducibility, it would be useful to reset the seeds of each imported library's random number generators (RNGs) when *GreatAI* is configured. Thus, a feature has been added to detect and reset RNGs of installed and imported libraries. This certainly will not solve the reproducibility crisis [82] on its own, however, in some cases, it can result in one fewer step to miss.

**Utility of LargeFiles** For the purposes of the documentation, the finetuning was conducted in the Google Colab online environment which is excellent for providing anyone with GPU-time for free. However, notebook environments are ephemeral, resulting in the need to manually upload and download all relevant data whenever a new virtual machine (VM) instance is granted. The `LargeFile` implementation alleviated this problem by handling the uploads and downloads automatically. Of course, first, backwards compatibility had to be solved for Python 3.7 which is the only available environment in Colab.

The best validation results were achieved after 8 epochs which was slightly more than expected but is presumably due to the weight decay. The confusion matrix on the test split can be seen in Figure 5.7: regardless the subjective definition of the task, SciBERT manages to achieve good quality which is indicated by an F1-score of **0.89**.

Let us check how well the selected sentences correspond with the tech-transfer potential. Users and in-house experts can rate publications (from a tech-transfer perspective) by assigning them to one of four categories: `A`, `B`, `C`, and `D` with `A` being the most and `D` the least promising. This feedback is stored and used for analytic and training purposes. Since both the feedback grade and the "highlights" are supposed to reflect the same aspect of papers, therefore, we can reasonably expect some correlation between them.

Figure 5.8 shows the ratio of summary candidate sentences as predicted by the finetuned model in 4 categories (grades) of papers. The two datasets come from non-overlapping sets of papers, hence, the results come solely from the model's ability to generalise. It is interesting to see that the Spearman's rank correlation coefficient [83] between the normalised "highlights" counts and the ratings of papers

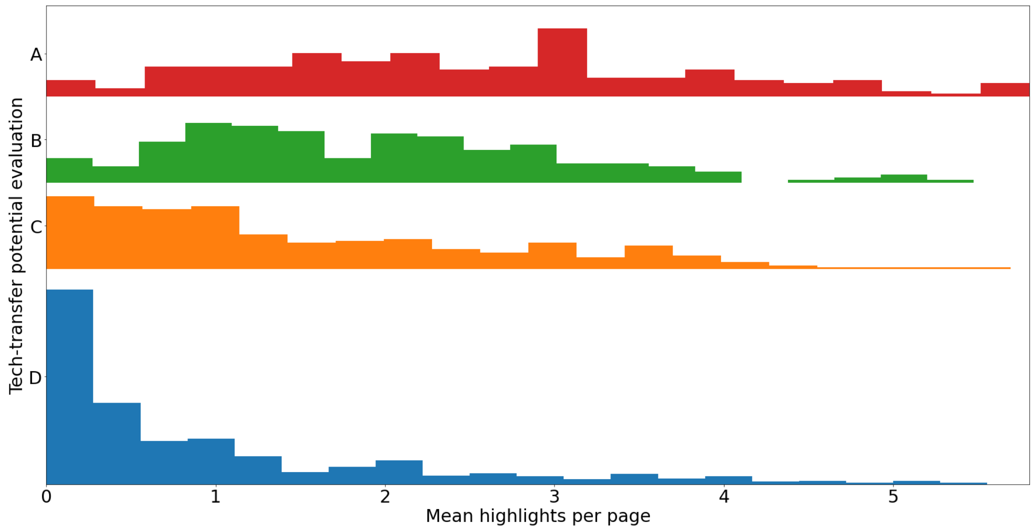



Figure 5.8: Distribution of mean predicted summary candidate sentence counts in 4 categories. Category `A` correspond to the most, while `D` to the least interesting papers based on mean user feedback. The sample size is 1406 (`D`=715, `C`=309, `B`=198, `A`=184). The histograms are on the same scale.

is **0.4784** and is statistically significant ($P = 5.4 \times 10^{-74}$). This proves the presence of a monotonic association. For context, the correlation between the grades and the number of sentences found by the baseline approach is 0.06597 ($P = 0.03$). We can conclude that the classifier's output is indicative of publications' tech-transfer potential.

### 5.3.4 Deployment

To implement the summarisation, at most the top 7 selected sentences are chosen as ranked by their log-probabilities. They are subsequently reordered according to their position in the text. As a quasi-explanation, the tokens' attention scores are visualised and overlaid on the highlighted sentences. The $i$-th token's visualised attention comes from summing up the attention weights of each of the last layer's heads between the `[CLS]` and the $i$-th token. To improve the end-user experience, a high-pass filter and a stop-word list is applied to the scores in order not to highlight the syntax-related tokens (punctuation, determiners). The service — after being integrated into the dashboard — can be seen in Figure 5.9.

**Design inspiration** In order to get insights into their inner workings, HuggingFace models can be given `output_attentions=True` in their constructor which results in a new property becoming accessible on the results for querying the attentions. The only issue with it is that it is a 5-dimensional matrix which makes exploring and understanding it non-obvious. In short, it has very low *Discoverability*. For example, the attentions for the UI are calculated with this expression:

```
np.sum(result.attentions[-1].numpy()[0], axis=0)[0][1:-1]
```

Even though the operation is conceptually simple, because of the opaque data structure, this is anything but obvious to comprehend. Therefore, it is clear that this needs to be avoided in my library design; it has to have an explicit and discoverable API which can be achieved by the use of typehints, descriptive property names, and docstrings.

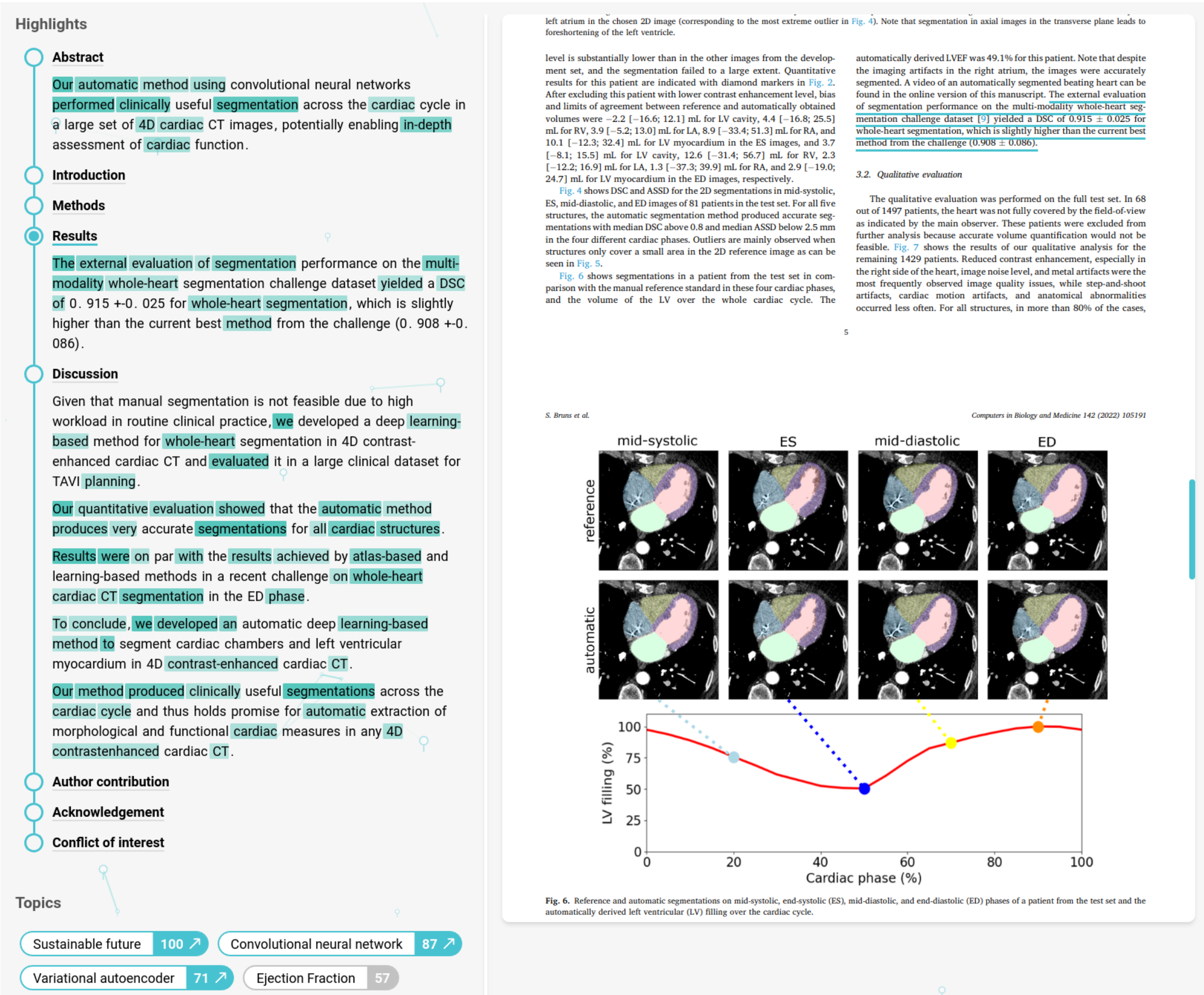


Figure 5.9: The tech-transfer summary of an academic publication ([84]). The titles and sentences can be clicked for navigating the paper on the right, meanwhile some explanation is provided by the highlighted words, the opacity of which corresponds to their attention weights.

## 5.4 Improving GreatAI

After having solved two problems by implementing two standalone services and integrating them into an existing ecosystem while relying on *GreatAI* as a primary tool, a wide variety of insights have been gained. In the next couple of subsections, the extra features and design decisions are presented that were motivated by the *Highlights service*. After which, the final surface of the API is described and evaluated by its relation to the SE4ML [4, 22] and AI engineering [23, 27] best practices.

### 5.4.1 Caching

Sustainability is an increasingly crucial concern of ethical AI [85]. Without discussing the pros and cons of the green computing movement [86], we can still agree that computing time should not be wasted. To this end, caching the results of expensive operations has to be considered in any AI deployment. In this case, the *Highlights service* is often called multiple times from different other services with the same parameters. With each operation taking up to a couple of seconds, implementing caching can lead to vastly faster response times and an overall more socially conscious deployment.

### 5.4.2 Revisiting `parallel_map`

Even though most inference functions are CPU-bound, turns out, sometimes they involve IO, especially, when relying on the results of other, remote models. This means that a significant performance improvement can be achieved by implementing some inference functions asynchronously [87]. Thus, *GreatAI* also has to support decorating both regular (synchronous) and asynchronous functions. There is one notable consequence of this: the batch processing feature also has to be compatible with `async` inference functions. Batch processing is still a useful feature since it is likely that async inference functions are both IO (remote calls) and CPU (local evaluation) constrained at the same time, thus, they can benefit from multi-core parallelisation.

However, the standard library's `multiprocessing`, the third party `multiprocess` [88], and, another popular library, `joblib`[17] all lack the support for efficiently parallelising async functions. For this reason, `parallel_map` is reimplemented to create an event-loop in each worker process to keep the efficiency of non-blocking IO while also providing parallelisation for the CPU-bound sections of code.

### 5.4.3 Programmatic integration

Apart from supporting `async` calls, there are a couple of more step that can be taken to help integrating any service with a *GreatAI* deployment. This is implemented by the `call_remote_great_ai` function which hides the networking required to call a *GreatAI* instance's REST API. It takes care of validation, automatic retries, serialisation, and deserialisation. This comes with the added benefit of encouraging decoupled services because the friction of integrating them is no longer noticeable which is beneficial for human collaboration [89].

Additionally, a REST API is generated with its accompanying OpenAPI schema[18] and served with a Swagger template. It also contains metadata about the function, for instance, its docstring, version, and version of its registered models concatenated in order to be SemVer[19] compatible. These can be seen in Figure 5.10. This, combined with a `/version` HTTP endpoint for programmatic access and

[17]joblib.readthedocs.io/en/latest

[18]swagger.io/specification

[19]semver.org

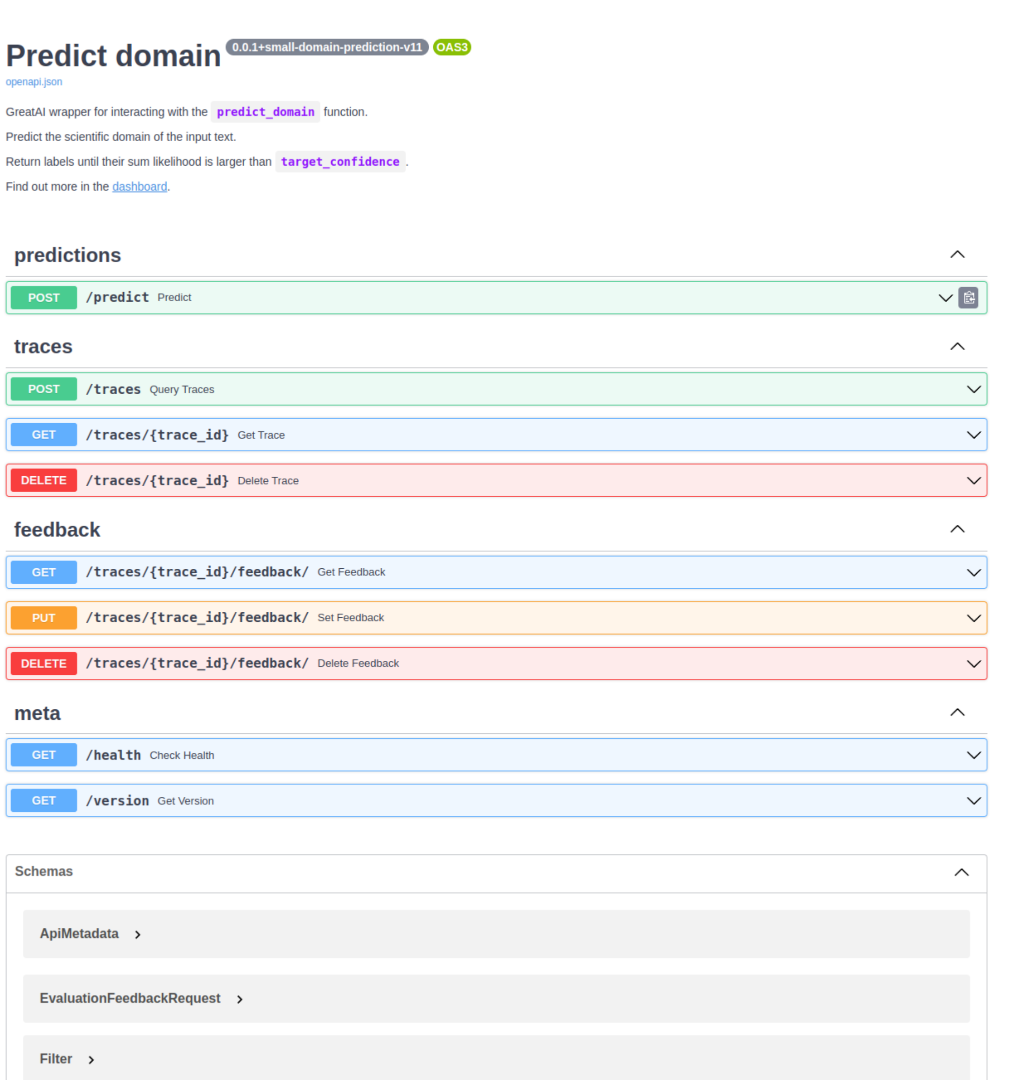



Figure 5.10: Documentation of the automatically scaffolded REST API of a *GreatAI* service. Notice, how its version string includes its registered models in a SemVer compliant way: `0.0.1+small-domain-prediction-v11`.

validation of the service's metadata proved to be key features when integrating the *Highlights service* into ScoutinScience's service-based architecture.

### 5.4.4 Human integration

Even though the REST API of *GreatAI* services exposes all necessary features[20] which are great for programmatic access, these are not ideal for direct human consumption. In order to ease the introduction of *GreatAI* services, a rudimentary dashboard is — optionally — generated next to the REST API. The dashboard's main features can be observed in Figures 5.11, 5.12, and 5.13. The diagrams and filterable/sortable table are interconnected and are automatically updated, the reactive behaviour is provided by the Dash framework [90].

[20]Such as providing feedback per prediction, complexly filtering and sorting traces, create-read-update-delete (CRUD) operations for the feedback and traces, accessing live monitoring info (current configuration, versions, cache statistics), etc.

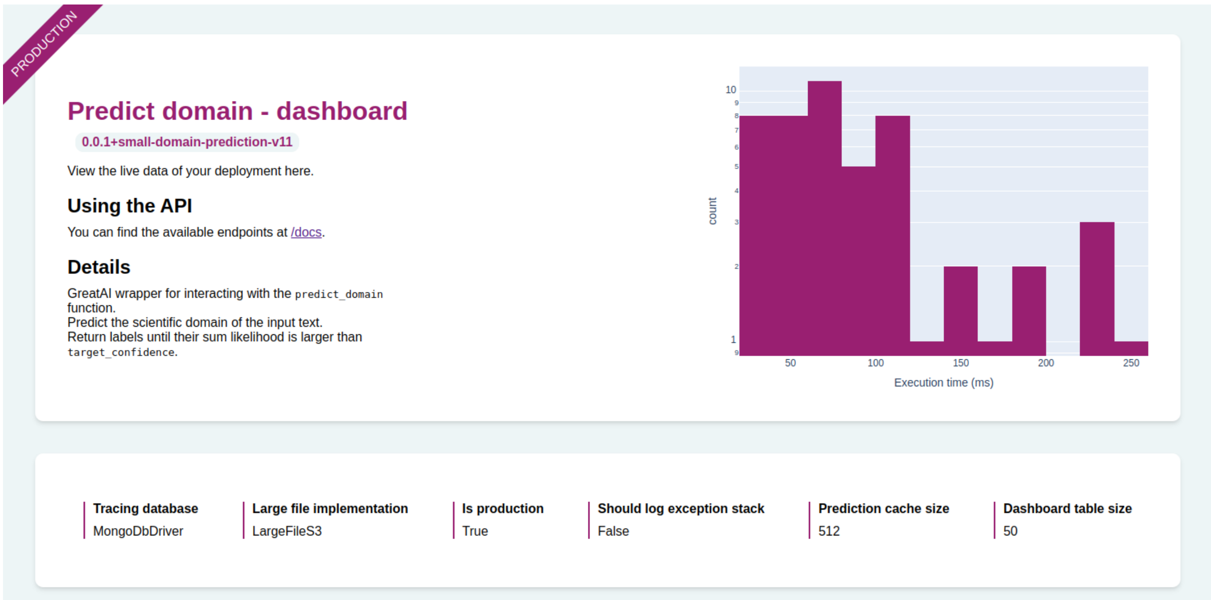
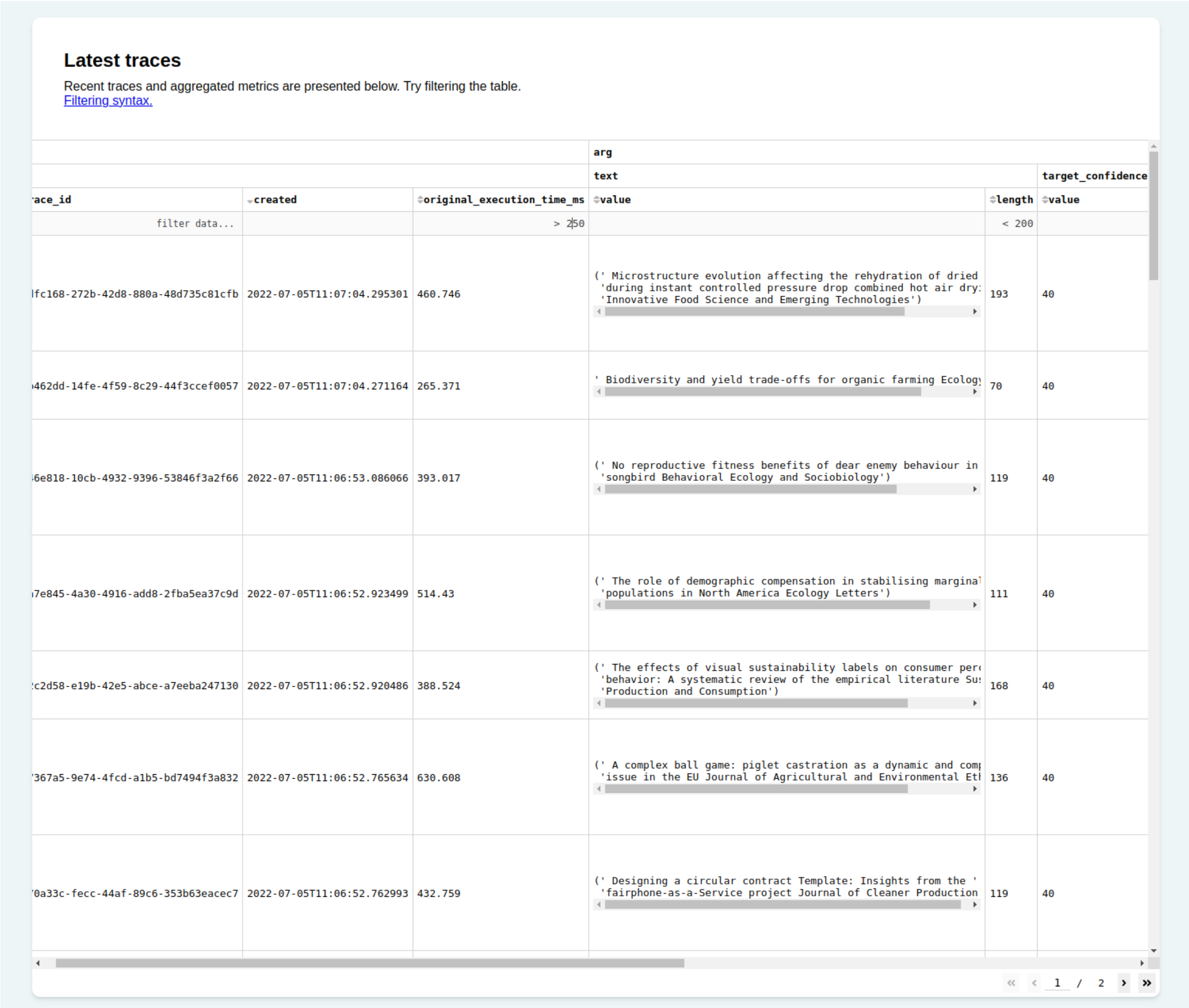



Figure 5.11: The header of the automatically generated dashboard of the service from Section 5.1. A documentation is generated on the left, while the histogram of response times is shown on the right. The current configuration is prominently displayed on the bottom.




| race_id | created | original_execution_time_ms | arg text value | arg text length | target_confidence value |
|---|---|---|---|---|---|
| filter data... | | > 250 | | < 200 | |
| lfc168-272b-42d8-880a-48d735c81cfb | 2022-07-05T11:07:04.295301 | 460.746 | (' Microstructure evolution affecting the rehydration of dried 'during instant controlled pressure drop combined hot air dry: 'Innovative Food Science and Emerging Technologies') | 193 | 40 |
| 462dd-14fe-4f59-8c29-44f3ccef0057 | 2022-07-05T11:07:04.271164 | 265.371 | ' Biodiversity and yield trade-offs for organic farming Ecolog | 70 | 40 |
| 6e818-10cb-4932-9396-53846f3a2f66 | 2022-07-05T11:06:53.086066 | 393.017 | (' No reproductive fitness benefits of dear enemy behaviour in 'songbird Behavioral Ecology and Sociobiology') | 119 | 40 |
| 7e845-4a30-4916-add8-2fba5ea37c9d | 2022-07-05T11:06:52.923499 | 514.43 | (' The role of demographic compensation in stabilising marginal 'populations in North America Ecology Letters') | 111 | 40 |
| c2d58-e19b-42e5-abce-a7eeba247130 | 2022-07-05T11:06:52.920486 | 388.524 | (' The effects of visual sustainability labels on consumer per 'behavior: A systematic review of the empirical literature Su: 'Production and Consumption') | 168 | 40 |
| 367a5-9e74-4fcd-a1b5-bd7494f3a832 | 2022-07-05T11:06:52.765634 | 630.608 | (' A complex ball game: piglet castration as a dynamic and com 'issue in the EU Journal of Agricultural and Environmental Et | 136 | 40 |
| 0a33c-fecc-44af-89c6-353b63eacec7 | 2022-07-05T11:06:52.762993 | 432.759 | (' Designing a circular contract Template: Insights from the ' 'fairphone-as-a-Service project Journal of Cleaner Production | 119 | 40 |

Figure 5.12: A dynamically updating, tabular view of traces matching a user-defined filter. Useful for exploring historical predictions or debugging the cause of exceptions (which are also searchable). The filters set in the table affect the other diagrams of the dashboard.

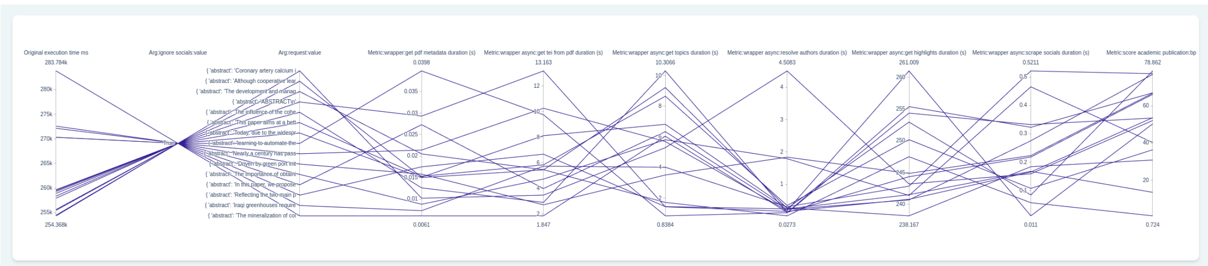


Figure 5.13: A parallel coordinates view of the traces displayed in the table above. Adding new axes is as easy as calling `log_metric` inside the inference function.

# Chapter 6

# Results

It should not be surprising that neither data scientists nor software engineers can be replaced by software libraries. However, a non-negligible subset of their processes can be partially or fully automated, especially when it comes to packaging and deploying AI/ML services. My goal was to design a library with an API that finds the balance between being simple enough to adopt without friction, yet useful/powerful enough to be adopted. Simplicity is subjective and it will be discussed separately in Section 6.1. For now, let us look at the utility of *GreatAI*.

For answering **RQ3** — *To what extent can GreatAI automatically implement AI deployment best practices?* — a comparison is presented in the following that illustrates which best-practices can be implemented/scaffolded/configured with little user input; hence, through a simple and streamlined API. Tables 6.1 and 6.2 summarise the implemented best practices in the context of practices found by prior surveys of scientific and grey literature [4, 22, 23].

In order to show an accurately nuanced representation, a *Level of support* is determined for each best practice on a scale of *Fully automated*, *Supported*, and *Partially supported*. For instance, *Use static analysis to check code quality* from Table 6.1 is *Supported* because the entire public interface of *GreatAI* is correctly typed (including generics and asynchronous coroutines) and compatible with `mypy` and `Pylance`. This means that when *GreatAI* is used in any Python project, these tools can be applied to statically check the soundness of the project's integration with *GreatAI*. However, if the library's user does not use typehints in their code and it contains more complex control flow, it can only be partially typechecked. In short, this best practice is supported, and a considerable part of it is already implemented by *GreatAI*, but clients should still keep in mind that they might also need to make effort to fully implement it.

This is not the case for *Log production predictions with the model's version and input data* because by default, it is automatically implemented when calling `@GreatAI.create`. Users can still specify the exact expected behaviour, e.g.: where to store traces, additional metrics to log, or disabling the logging of sensitive input. Nevertheless, without input from the library's user, the best practice is already reasonably well implemented.

In Table 6.2, six additional best practices have been added which are generally well-known software engineering considerations that are also applicable to AI/ML deployments. These have not explicitly made it into the aforementioned surveys, however according to the insights gained from Sections 5.1 and 5.3, implementing them has a positive effect on deployment quality. In future research, attention could be given to their level of industry-wide adoption and quantitative utility.

Table 6.1: A subset of AI lifecycle best practices and the level of support *GreatAI* provides for them. The level of support is one of *Fully automated* (✓✓) which means that no action is required from the user, *Supported* (✓) only automates the reasonably automatable aspects, while *Partially supported* (∼) provides some useful features but the client is expected to build on top of these.

| **Best practice** | **Implementation** | **Support** |
|---|---|---|
| Use sanity checks for all external data sources[1] | `@parameter` | ✓ |
| Check that input data is complete, balanced, and well distributed[1] | `@parameter` | ∼ |
| Write reusable scripts for data cleaning and merging (for NLP)[1] | `utilities` | ✓✓ |
| Make data sets available on shared infrastructure[1] | `large_file` | ✓✓ |
| Test all feature extraction code (for NLP)[1] | `utilities` | ✓✓ |
| Employ interpretable models when possible[1] | `views` | ∼ |
| Continuously measure model quality and performance[1, 2] | Feedback API | ✓ |
| Use versioning for data, model, configurations and training scripts[1, 2] | `@use_model`, versioning | ✓✓ |
| Run automated regression tests[1] | `*_ground_truth` | ✓ |
| Use continuous integration[1] | Docker Image, WSGI application | ✓ |
| Use static analysis to check code quality[1] | Fully typed API with generics | ✓ |
| Assure application security[1] | Code is automatically audited | ∼ |
| Automate model deployment, enable shadow deployment[1, 2] | Docker Image & scripts | ✓ |
| Enable automatic rollbacks for production models[1, 2] | Docker Image & scripts | ∼ |
| Continuously monitor the behaviour of deployed models[1, 2] | Dashboard, metrics endpoints | ✓✓ |
| Log production predictions with the model's version and input data[1] | `@GreatAI.create` | ✓✓ |

[1] SE4ML best practices from Table 2 of [4], and Table 1 of [22].

[2] Reported state-of-the-art and state-of-practice practices from Tables 2, 3, and 4 of [23].

Table 6.2: A subset of AI lifecycle best practices and the level of support *GreatAI* provides for them. The level of support is one of *Fully automated* (✓✓) which means that no action is required from the user, *Supported* (✓) only automates the reasonably automatable aspects, while *Partially supported* (∼) provides some useful features but the client is expected to build on top of these.

| **Best practice** | **Implementation** | **Support** |
|---|---|---|
| Execute validation techniques: error rates and cross-validation[2] | `*_ground_truth` | ✓ |
| Store models in a single format for ease of use[2] | `save_model` | ✓✓ |
| Rewrite from data analysis to industrial development language[2] | Jupyter Notebook deployment | ✓ |
| Equip with web interface, package image, provide REST API[2] | `@GreatAI.create` | ✓✓ |
| Provide simple API for serving batch and real-time requests[2] | `@GreatAI.create` | ✓✓ |
| For reproducibility, use standard runtime and configuration files[2] | `utilities.ConfigFile`, Dockerfile | ✓ |
| Integration with existing data infrastructure[2] | GridFS, S3 support | ✓✓ |
| Select ML solution fully integrated with databases[2] | MongoDB, PostgreSQL support | ✓✓ |
| Querying, visualising and understanding metrics and event logging[2] | Dashboard, Traces API | ✓✓ |
| Measure accuracy of deployed model to ensure data drifts are noticed[2] | Feedback API | ✓ |
| Apply automation to trigger model retraining[2] | Feedback API | ∼ |
| Allow experimentation with the inference code[3] | Development mode & auto-reload | ✓✓ |
| Keep the model's API and documentation together[3] | Dashboard and Swagger | ✓✓ |
| Parallelise feature extraction[3] | `parallel_map` | ✓✓ |
| Cache predictions[3] | `@GreatAI.create` | ✓✓ |
| Async support for top-down chaining models[3] | All decorators support async | ✓✓ |
| Common schemas for common prediction tasks[3] | `views` | ✓ |

[2] Reported state-of-the-art and state-of-practice practices from Tables 2, 3, and 4 of [23].

[3] Additional software engineering best practices applicable to AI/ML deployments encountered while designing and using *GreatAI*.

Quantifying the number of implemented best practices would be misleading since their scope and importance cover a wide — sometimes overlapping — range. Especially because there is some overlap between the different reviews and even within the reviews. However, it is still clear that a large number of best practices can be given a *Fully automated* implementation by *GreatAI*'s design while and even larger number of them can be augmented by the library. This proves the feasibility of designing simple APIs using the techniques of Chapter 4 for decreasing the complexity of correctly deploying AI services (**RQ2**).

## 6.1 Interviews

Let us finally tackle the question of generalisability using the interview methodology described in Section 3.2.

One of the takeaways of Chapter 2 was that Seldon Core is useful for implementing or helping to implement most best practices. Nonetheless, it also has an initial threshold that must be surmounted before implementing even a single best practice. According to the adoption rate surveys, this discourages a large portion of practitioners from using it or other similar frameworks. *GreatAI* offers a different mix of features, the initial threshold is virtually non-existent: best practices can be immediately applied. But at the same time, the presented solution covers a smaller number of practices. The hypothesis is that the latter approach aligns better with the expectations of professionals.

### 6.1.1 Threats to validity

## 6.2 Future work

The primary purpose of *GreatAI* was to serve as a proxy through which its design decisions could be tested and evaluated in their practical context. For this reason, its design aimed to be a proof-of-principle for validating hypotheses and answering research questions. After successfully doing that, it has been turned into a practical software library suitable for production-use[1]. Although it has already proved its utility, it has also shown that extending its functionality would be worthwhile. Therefore, a number of potential improvements to *GreatAI* are presented below.

### 6.2.1 More data science

The cases presented in Chapter 5 revolved around NLP. This, unsurprisingly, heavily influenced the design process. The two most notable effects can be found in the REST API's `/predict` endpoint and some `utilities` functions. The former is streamlined to accept JSON compatible data while the latter gives robust feature extraction support for only textual inputs. Supporting the easy, direct upload of larger non-JSON files — e.g. by saving them to S3 and showing a preview for them on the Dashboard's trace table — and extending `utilities` to handle multimedia formats should be sufficient for widely extending the scope of applicability of *GreatAI*.

### 6.2.2 More software engineering

In order to greatly simplify its API, each *GreatAI* Trace is a single document with a well-defined, multi-level schema that clients can also extend by calling `log_metric`. MongoDB provides a convenient (and popular) method for persisting such documents, however, if there is some existing database in the envi-

[1] pypi.org/project/great-ai

ronment, storing Traces in that can be favourable. PostgreSQL is a popular choice and it also features good JSON document support. Hence, introducing first-class integration for PostgreSQL could benefit some clients.

As described in Designing Data-intensive Applications [53], services can fall into three broad categories: online systems, batch processing, and stream processing (near-teal-time systems). As of yet, *GreatAI* only provides streamlined support for the first two. Thus, developer experience could be improved by providing simple, direct integration with popular message queues/protocols, such as Apache Kafka [91], AWS SQS [92], or AMQP [93].

Some metrics of *GreatAI*, such as the cache statistics, versions, and derived data from traces can be already conveniently queried from its REST API. Nevertheless, adding support for the de facto standard metric gathering tool Prometheus could save the library's users from one more integration steps.

The common theme among the above-mentioned opportunities is that they could be reasonably well implemented without any user input, making them inline with the library's philosophy. Of course, the open-source nature of *GreatAI* also allows anyone to already provide support for a wide range of integrations. Additionally, the scope could be also reasonably extended, i.e. more practices could be covered by including more criteria next to the GREAT ones.

# Chapter 7

# Conclusion

todo

*GreatAI* may have the potential to bridge the gap between data science and software engineering. Stemming from the bidirectional nature of bridges, we can look at the framework from two perspectives: for professionals closer to the field of data science, it provides an automatic scaffolding of software facilities that are required for deploying, monitoring, and iterating on their models. For software engineers, it highlights the necessary steps required for robust and improvable deployments — while at the same time — saves them from the menial work of implementing these constructs manually. While most importantly, it proves that increasing the adoption rate of AI/ML deployment best practices is viable by designing narrower and deeper APIs.

Good deployments benefit all of us. Continued research into the means of good deployments remains crucial. However, next to that — as the presented results show — better deployments can be also achieved by facilitating the *transition* step of the AI lifecycle. Having automated implementations, even if for just simpler best practices, leaves professionals more time to tackle other deployment challenges and less opportunities to miss crucial steps. Overall, resulting in more implemented practices, hence, robust and trustworthy production software.

## 7.1 Concluding remarks

# References


[1] Wirtz B. W., Weyerer J. C., Geyer C.: Artificial intelligence and the public sector—applications and challenges. International Journal of Public Administration: vol. 42(7), pp. 596–615 (2019)

[2] Bosch J., Olsson H. H., Crnkovic I.: Engineering ai systems: A research agenda. In Artificial Intelligence Paradigms for Smart Cyber-Physical Systems: pp. 1–19: IGI global (2021)

[3] Jordan M. I., Mitchell T. M.: Machine learning: Trends, perspectives, and prospects. Science: vol. 349(6245), pp. 255–260 (2015)

[4] Serban A., van der Blom K., Hoos H., Visser J.: Adoption and effects of software engineering best practices in machine learning. In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM): pp. 1–12 (2020)

[5] O'neil C.: Weapons of math destruction: How big data increases inequality and threatens democracy. Broadway Books (2016)

[6] Haakman M., Cruz L., Huijgens H., van Deursen A.: AI lifecycle models need to be revised. Empirical Software Engineering: vol. 26(5), pp. 1–29 (2021)

[7] Amershi S., Begel A., Bird C., DeLine R., Gall H., Kamar E., Nagappan N., Nushi B., Zimmermann T.: Software engineering for machine learning: A case study. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP): pp. 291–300: IEEE (2019)

[8] de Souza Nascimento E., Ahmed I., Oliveira E., Palheta M. P., Steinmacher I., Conte T.: Understanding development process of machine learning systems: Challenges and solutions. In 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM): pp. 1–6: IEEE (2019)

[9] Sculley D., Holt G., Golovin D., Davydov E., Phillips T., Ebner D., Chaudhary V., Young M., Crespo J. F., Dennison D.: Hidden technical debt in machine learning systems. Advances in neural information processing systems: vol. 28 (2015)

[10] Wieringa R. J.: Design science methodology for information systems and software engineering. Springer (2014)

[11] Raji I. D., Smart A., White R. N., Mitchell M., Gebru T., Hutchinson B., Smith-Loud J., Theron D., Barnes P.: Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In Proceedings of the 2020 conference on fairness, accountability, and transparency: pp. 33–44 (2020)

[12] Akbik A., Bergmann T., Blythe D., Rasul K., Schweter S., Vollgraf R.: FLAIR: An easy-to-use framework for state-of-the-art NLP. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations): pp. 54–59 (2019)

[13] Wolf T., Debut L., Sanh V., Chaumond J., Delangue C., Moi A., Cistac P., Rault T., Louf R., Funtowicz M., et al.: Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771 (2019)

[14] Devlin J., Chang M. W., Lee K., Toutanova K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[15] Srinivasa-Desikan B.: Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras. Packt Publishing Ltd (2018)

[16] Rehrek R., Sojka P., et al.: Gensim—statistical semantics in python. Retrieved from genism. org (2011)

[17] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., et al.: Scikit-learn: Machine learning in Python. the Journal of machine Learning research: vol. 12, pp. 2825–2830 (2011)

[18] Chen T., Guestrin C.: XGBoost. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining: ACM: doi:10.1145/2939672.2939785: URL https://doi.org/10.1145%2F2939672.2939785 (2016)

[19] Sun Y., Agostini N. B., Dong S., Kaeli D.: Summarizing CPU and GPU design trends with product data. arXiv preprint arXiv:1911.11313 (2019)

[20] Food, Administration D., et al.: Proposed regulatory framework for modifications to artificial intelligence/machine learning (AI/ML)-based software as a medical device (SaMD). APO (2019)

[21] Thiée L. W.: A systematic literature review of machine learning canvases. INFORMATIK 2021 (2021)

[22] Serban A., van der Blom K., Hoos H., Visser J.: Practices for engineering trustworthy machine learning applications. In 2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN): pp. 97–100: IEEE (2021)

[23] John M. M., Holmström Olsson H., Bosch J.: Architecting AI Deployment: A Systematic Review of State-of-the-art and State-of-practice Literature. In International Conference on Software Business: pp. 14–29: Springer (2020)

[24] Li L. E., Chen E., Hermann J., Zhang P., Wang L.: Scaling machine learning as a service. In International Conference on Predictive Applications and APIs: pp. 14–29: PMLR (2017)

[25] Wang D., Ram P., Weidele D. K. I., Liu S., Muller M., Weisz J. D., Valente A., Chaudhary A., Torres D., Samulowitz H., et al.: Autoai: Automating the end-to-end ai lifecycle with humans-in-the-loop. In Proceedings of the 25th International Conference on Intelligent User Interfaces Companion: pp. 77–78 (2020)

[26] Joshi A. V.: Amazon's machine learning toolkit: Sagemaker. In Machine Learning and Artificial Intelligence: pp. 233–243: Springer (2020)

[27] John M. M., Olsson H. H., Bosch J.: AI Deployment Architecture: Multi-Case Study for Key Factor Identification. In 2020 27th Asia-Pacific Software Engineering Conference (APSEC): pp. 395–404: IEEE (2020)

[28] Baylor D., Breck E., Cheng H. T., Fiedel N., Foo C. Y., Haque Z., Haykal S., Ispir M., Jain V., Koc L., et al.: Tfx: A tensorflow-based production-scale machine learning platform. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining: pp. 1387–1395 (2017)

[29] Moritz P., Nishihara R., Wang S., Tumanov A., Liaw R., Liang E., Elibol M., Yang Z., Paul W., Jordan M. I., et al.: Ray: A distributed framework for emerging {AI} applications. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18): pp. 561–577 (2018)

[30] Klaise J., Van Looveren A., Vacanti G., Coca A.: Alibi Explain: algorithms for explaining machine learning models. Journal of Machine Learning Research: vol. 22(181), pp. 1–7 (2021)

[31] Prado M. D., Su J., Saeed R., Keller L., Vallez N., Anderson A., Gregg D., Benini L., Llewellynn T., Ouerhani N., et al.: Bonseyes AI pipeline—Bringing AI to you: End-to-end integration of data, algorithms, and deployment tools. ACM Transactions on Internet of Things: vol. 1(4), pp. 1–25 (2020)

[32] Wirth R., Hipp J.: CRISP-DM: Towards a standard process model for data mining. In Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining: vol. 1: pp. 29–40: Manchester (2000)

[33] Davison R., Martinsons M. G., Kock N.: Principles of canonical action research. Information systems journal: vol. 14(1), pp. 65–86 (2004)

[34] Shull F., Singer J., Sjøberg D. I.: Guide to advanced empirical software engineering. Springer (2007)

[35] Halcomb E. J., Davidson P. M.: Is verbatim transcription of interview data always necessary? Applied nursing research: vol. 19(1), pp. 38–42 (2006)

[36] Fereday J., Muir-Cochrane E.: Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. International journal of qualitative methods: vol. 5(1), pp. 80–92 (2006)

[37] Cruz L., Abreu R.: Catalog of energy patterns for mobile applications. Empirical Software Engineering: vol. 24(4), pp. 2209–2235 (2019)

[38] Davis F. D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly: pp. 319–340 (1989)

[39] Marangunić N., Granić A.: Technology acceptance model: a literature review from 1986 to 2013. Universal access in the information society: vol. 14(1), pp. 81–95 (2015)

[40] Riemenschneider C. K., Hardgrave B. C., Davis F. D.: Explaining software developer acceptance of methodologies: a comparison of five theoretical models. IEEE transactions on Software Engineering: vol. 28(12), pp. 1135–1145 (2002)

[41] Wu C. S., Cheng F. F., Yen D. C., Huang Y. W.: User acceptance of wireless technology in organi-

zations: A comparison of alternative models. Computer Standards & Interfaces: vol. 33(1), pp. 50–58 (2011)
[42] Bland J. M., Altman D. G.: Statistics notes: Cronbach's alpha. Bmj: vol. 314(7080), p. 572 (1997)
[43] Hynes N., Sculley D., Terry M.: The data linter: Lightweight, automated sanity checking for ml data sets. In NIPS MLSys Workshop: vol. 1 (2017)
[44] Bishop M.: Robust programming. handout for (1998)
[45] Ritchie D. M., Thompson K.: The UNIX timesharing system. Bell System Technical Journal: vol. 57(6), pp. 1905–1929 (1978)
[46] Salus P. H.: A quarter century of UNIX. ACM Press/Addison-Wesley Publishing Co. (1994)
[47] Ousterhout J. K.: A philosophy of software design: vol. 98. Yaknyam Press Palo Alto (2018)
[48] Hermans F.: The Programmer's Brain: What every programmer needs to know about cognition. Simon and Schuster (2021)
[49] Blackwell A. F., Britton C., Cox A., Green T. R., Gurr C., Kadoda G., Kutar M. S., Loomes M., Nehaniv C. L., Petre M., et al.: Cognitive dimensions of notations: Design tools for cognitive technology. In International conference on cognitive technology: pp. 325–341: Springer (2001)
[50] Arnaoudova V., Di Penta M., Antoniol G.: Linguistic antipatterns: What they are and how developers perceive them. Empirical Software Engineering: vol. 21(1), pp. 104–158 (2016)
[51] Deissenboeck F., Pizka M.: Concise and consistent naming. Software Quality Journal: vol. 14(3), pp. 261–282 (2006)
[52] Butler S., Wermelinger M., Yu Y., Sharp H.: Relating identifier naming flaws and code quality: An empirical study. In 2009 16th Working Conference on Reverse Engineering: pp. 31–35: IEEE (2009)
[53] Kleppmann M.: Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc." (2017)
[54] Procida D.: Diátaxis documentation framework. URL https://diataxis.fr/
[55] Miller G. A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review: vol. 63(2), p. 81 (1956)
[56] Martin R. C.: Clean code: a handbook of agile software craftsmanship. Pearson Education (2009)
[57] Maron M. E.: Automatic indexing: an experimental inquiry. Journal of the ACM (JACM): vol. 8(3), pp. 404–417 (1961)
[58] Rivest M., Vignola-Gagné E., Archambault É.: level classification of scientific publications: A comparison of deep learning, direct citation and bibliographic coupling. PloS one: vol. 16(5), p. e0251493 (2021)
[59] Beltagy I., Lo K., Cohan A.: SciBERT: A pretrained language model for scientific text. arXiv preprint arXiv:1903.10676 (2019)
[60] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I.: Attention is all you need. Advances in neural information processing systems: vol. 30 (2017)
[61] Lo K., Wang L. L., Neumann M., Kinney R. M., Weld D. S.: S2ORC: The Semantic Scholar Open Research Corpus. In ACL (2020)
[62] Wang K., Shen Z., Huang C., Wu C. H., Eide D., Dong Y., Qian J., Kanakia A., Chen A., Rogahn R.: A review of microsoft academic services for science of science studies. Frontiers in Big Data: vol. 2, p. 45 (2019)
[63] Buckley C.: Implementation of the SMART information retrieval system. Tech. rep.: Cornell University (1985)
[64] Hand D. J., Yu K.: Idiot's Bayes—not so stupid after all? International statistical review: vol. 69(3), pp. 385–398 (2001)
[65] Van Der Weide T., Papadopoulos D., Smirnov O., Zielinski M., Van Kasteren T.: Versioning for end-to-end machine learning pipelines. In Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning: pp. 1–9 (2017)
[66] Leite L., Rocha C., Kon F., Milojicic D., Meirelles P.: A survey of DevOps concepts and challenges. ACM Computing Surveys (CSUR): vol. 52(6), pp. 1–35 (2019)
[67] Gamma E., Helm R., Johnson R., Johnson R. E., Vlissides J., et al.: Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH (1995)
[68] Mikolov T., Chen K., Corrado G., Dean J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
[69] El-Kassas W. S., Salama C. R., Rafea A. A., Mohamed H. K.: Automatic text summarization: A comprehensive survey. Expert Systems with Applications: vol. 165, p. 113679 (2021)
[70] Maynez J., Narayan S., Bohnet B., McDonald R.: On faithfulness and factuality in abstractive summarization. arXiv preprint arXiv:2005.00661 (2020)
[71] Berkovsky S., Baldwin T., Zukerman I.: Aspect-based personalized text summarization. In In-

ternational Conference on Adaptive Hypermedia and Adaptive Web-Based Systems: pp. 267–270: Springer (2008)

[72] Hayashi H., Budania P., Wang P., Ackerson C., Neervannan R., Neubig G.: WikiAsp: A dataset for multi-domain aspect-based summarization. Transactions of the Association for Computational Linguistics: vol. 9, pp. 211–225 (2021)

[73] Iwatsuki K., Boudin F., Aizawa A.: An evaluation dataset for identifying communicative functions of sentences in English scholarly papers. In 12th Conference on Language Resources and Evaluation (LREC 2020) (2020)

[74] Iwatsuki K., Aizawa A.: Extraction of Formulaic Expressions from Scientific Papers. methods: vol. 362, pp. 7–469 (2021)

[75] Iwatsuki K., Aizawa A.: Communicative-Function-Based Sentence Classification for Construction of an Academic Formulaic Expression Database. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume: pp. 3476–3497 (2021)

[76] Iwatsuki K., Boudin F., Aizawa A.: Extraction and evaluation of formulaic expressions used in scholarly papers. Expert Systems with Applications: vol. 187, p. 115840 (2022)

[77] Schütze H., Manning C. D., Raghavan P.: Introduction to information retrieval: vol. 39. Cambridge University Press Cambridge (2008)

[78] Cohen J.: A coefficient of agreement for nominal scales. Educational and psychological measurement: vol. 20(1), pp. 37–46 (1960)

[79] McHugh M. L.: Interrater reliability: the kappa statistic. Biochemia medica: vol. 22(3), pp. 276–282 (2012)

[80] Verberne S., Krahmer E., Hendrickx I., Wubben S., van Den Bosch A.: Creating a reference data set for the summarization of discussion forum threads. Language Resources and Evaluation: vol. 52(2), pp. 461–483 (2018)

[81] Loshchilov I., Hutter F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)

[82] Hutson M.: Artificial intelligence faces reproducibility crisis (2018)

[83] Spearman C.: The proof and measurement of association between two things. The American Journal of Psychology (1961)

[84] Bruns S., Wolterink J. M., van den Boogert T. P., Runge J. H., Bouma B. J., Henriques J. P., Baan J., Viergever M. A., Planken R. N., Išgum I.: Deep learning-based whole-heart segmentation in 4D contrast-enhanced cardiac CT. Computers in biology and medicine: vol. 142, p. 105191 (2022)

[85] van Wynsberghe A.: Sustainable AI: AI for sustainability and the sustainability of AI. AI and Ethics: vol. 1(3), pp. 213–218 (2021)

[86] Kurp P.: Green Computing. Commun. ACM: vol. 51(10), p. 11–13: ISSN 0001-0782: doi:10.1145/1400181.1400186: URL https://doi.org/10.1145/1400181.1400186 (2008)

[87] Tilkov S., Vinoski S.: Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing: vol. 14(6), pp. 80–83 (2010)

[88] McKerns M. M., Strand L., Sullivan T., Fang A., Aivazis M. A.: Building a framework for predictive science. arXiv preprint arXiv:1202.1056 (2012)

[89] Hasselbring W.: Component-based software engineering. In Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies: pp. 289–305: World Scientific (2002)

[90] Shammamah Hossain: Visualization of Bioinformatics Data with Dash Bio. In Chris Calloway, David Lippa, Dillon Niederhut, David Shupe (editors), Proceedings of the 18th Python in Science Conference: pp. 126 – 133: doi:10.25080/Majora-7ddc1dd1-012 (2019)

[91] Kreps J., Narkhede N., Rao J., et al.: Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB: vol. 11: pp. 1–7 (2011)

[92] Garfinkel S.: An evaluation of Amazon's grid computing services: EC2, S3, and SQS. Harvard Computer Science Group Technical Report (2007)

[93] Vinoski S.: Advanced message queuing protocol. IEEE Internet Computing: vol. 10(6), pp. 87–89 (2006)

# Appendix A

# Best practices assessment

Similar to the approach of [4], participants are asked about their team's level of AI/ML deployment best practices adoption. The questions come from the entries of Tables 6.1 and 6.2 where *GreatAI* was determined to provide a support level of *Fully automated*.

**How well did the previous AI deployment that you have collaborated on implemented the following best-practices?** *Each statement can be rated on a 5-point Likert scale or as "Not applicable".*

1. Write reusable scripts for data cleaning and merging
2. Make data sets available on shared infrastructure
3. Use versioning for data, model, configurations and training scripts
4. Continuously monitor the behaviour of deployed models
5. Log production predictions with the model's version and input data
6. Store models in a single format for ease of use
7. Equip with web interface, package image, provide REST API
8. Provide simple API for serving batch and real-time requests
9. Integration with existing data infrastructure
10. Querying, visualising and understanding metrics and event logging
11. Allow experimentation with the inference code
12. Keep the model's API and documentation together
13. Parallelise feature extraction
14. Cache predictions
15. Async support for top-down chaining models

# Appendix B

# Technology acceptance model questionnaire

Following the methodology for parsimonious TAM of Wu et al. [41], each statement can be rated on a 7-point Likert scale.

**Perceived usefulness (PU)**

1. I believe the use of *GreatAI* improves the quality of AI deployments.
2. I believe the use of *GreatAI* would increase my productivity.
3. I believe the use of *GreatAI* can lead to robust and trustworthy deployments.
4. Overall, I found *GreatAI* useful when working with AI.

**Perceived ease of use (PEOU)**

1. I found the *GreatAI* easy to learn.
2. I found it is easy to employ *GreatAI* in practice.
3. I found it is easy to integrate *GreatAI* into an existing project.
4. Overall, I found *GreatAI* easy to use.

**Intention to use (ITU)**

1. Assuming *GreatAI* is applicable to my task, I predict that I will use it on a regular basis in the future.
2. Overall, I intend to use the *GreatAI* in my personal or professional projects.