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:	30/09/2022
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 libraries that accessibly expose state-of-the-art models. However, the transition from prototypes to production-ready AI applications is still a source of struggle across the industry. Even though professionals already have access to frameworks for deploying AI, case studies and developer surveys have found that many deployments do not follow best practices.

Objective: This thesis investigates the causes of and presents a possible solution to the asymmetry between the adoption of libraries for *applying* and those for *deploying* AI. The potential solution is validated through designing a software framework called *GreatAI*, which aims to facilitate General Robust End-to-end Automated Trustworthy deployments while attempting to overcome the practical drawbacks of earlier similar tools, e.g., *Seldon Core*, *AWS SageMaker*, and *TensorFlow Extended*.

Methods: *GreatAI* serves as a proxy for exploring the proposed design decisions; moreover, its initial focus is limited to the domain of natural language processing (NLP). Its design is created by applying the principles of design science methodology through iteratively shaping it in two case studies of a commercial NLP pipeline. Subsequently, interviews are conducted with ten practitioners to assess its applicability and generalisability.

Results: *GreatAI* helps implement 33 best practices through an accessible interface. These target the transition between the prototype and production phases of the AI development lifecycle. Feedback from professional data scientists and software engineers showed that ease of use and functionality are equally important in deciding to adopt deployment technologies, and the proposed framework was rated positively in both dimensions.

Conclusions: Increasing the overall maturity of industrial AI deployments by devising APIs with ease of adoption in mind is proved to be feasible. While *GreatAI* mainly focuses on NLP, the results show that the development and deployment of trustworthy AI services, in general, can be assisted by frameworks prioritising easy adoption while still streamlining the implementation of various best practices.

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 Ádám Kovács, Balázs Csomor, Bálint Bakcsa, Bendegúz Bendicsek, Joana Trashlieva, Leonardo Pohl, Lion Cassens, László Radnai, Máté Wolf, and Olivér Angyal for participating and providing insightful feedback in our interviews.

Chapter 1

Introduction

Artificial intelligence (AI) 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, an 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 radically lowers the barrier of entry for applying AI — and with that — can help use cases in various areas.

However, to achieve robust deployments, the successful integration of AI components into production-ready applications demands strong engineering methods [4]. That is why it is as essential as ever to also focus on the quality 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 does not follow best practices [4, 6, 7, 8, 9]. Utilising state-of-the-art machine learning (ML) models has become reasonably simple; applying them correctly is as intricate and nuanced as ever.

This thesis sets 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 to achieve some level of automation and maturity in their 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, we design a software framework called *GreatAI* and present it 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 a more 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 General Robust End-to-end Automated, and Trustworthy AI deployments.

The utility of *GreatAI* is examined and refined using the principles of design science methodology [10] through iteratively designing its API and implementation in two case studies concerning the natural language processing (NLP) pipeline of a commercial product in collaboration with ScoutinScience B.V. The goal of the aforementioned software suite is to evaluate technology-transfer opportunities in scientific publications. Subsequently, interviews are conducted with practitioners to validate the broader applicability and generalisability of the design.

The choice of case study subject is no coincidence; while working on the ScoutinScience Platform for the last two years, my colleagues and I have increasingly noticed the same recurring challenges in deploying and operating AI/ML pipelines. This has motivated me to pursue a general solution. Considering that the company's predominant field is NLP, the case studies, and hence, the prototype of *GreatAI* will also focus primarily on deploying NLP models. Nonetheless, the motivation for creating a general solution for all AI/ML contexts remains and will be taken into account every step of the way.

1.1 Research questions

We hypothesise that facilitating the adoption of AI deployment best practices is viable by finding less complex framework[1] designs that are easier to adopt in order to decrease the negative externality of misused AI. This paper investigates the hypothesis by answering the following research questions.

RQ1. To what extent does the complexity of deploying AI hinder 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 also be 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 that ought to be taken 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.

1.2 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 studies and the framework's interaction with them. 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.

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

Chapter 2

Background

Despite the long-standing history of artificial intelligence, industry awareness and adoption have only recently started to catch up meaningfully [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, clear advances are 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/ML[3] architectures 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 easily. In recent years, there has been a proliferation of highly accessible AI-libraries, many of which provide reusable models. For example, let us consider the domain of natural language processing. There are various options for finding AI solutions that work out of the box: FLAIR [14] and Hugging Face's transformers [15] let developers access state-of-the-art models and methods in only a couple of lines of code (in many cases 2 or 3). Using transfer-learning, Hugging Face enables its users to leverage vast amounts of knowledge learned by pretrained models (such as BERT [16] 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 libraries, the list of readily available solutions is vast: SpaCy [17], Gensim [18], scikit-learn [19], and XGBoost [20] 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. This, combined with the exponentially increasing computing power affordably available to consumers and businesses alike [21], results in AI that is accessible by many.

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

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

[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 [12]. ML is a well-defined subdomain of AI. However, most modern AI applications are also ML applications [13], hence, conflating the two terms may be slightly imprecise but usually not wrong.

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 already been thoroughly investigated.

When looking at AI/ML 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, 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-customise) dashboards for monitoring deployed models, resulting 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 a 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 explained: some steps are left out from time to time because they are forgotten. Similarly, Thiée [22] 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.

Serban et al. [4, 23] 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. 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%. Furthermore, Serban et al., in [23], identified another 14 best practices that concern trustworthy AI, mainly through data governance. They strove to complement high-level checklists with actionable best practices. Analysing 42 survey responses revealed a familiar pattern: most best practices had less than 50% adoption.

John et al. [24] compared and contrasted recent scientific and grey literature on AI deployments from which they extracted concrete challenges and practices. They also observed that most companies are placing many more models into production than in previous years. Additionally, they pointed out that numerous deployment techniques are absent from 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 [24], 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

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 [25]; they built a highly sophisticated infrastructure using techniques from distributed and high-performance computing. Though the authors note that this solution still has shortcomings in the form of rigidity (number of supported libraries and model types), it also allows for the easy extension of the system. Given the nature of the concerns and the amount of available resources, it is not surprising that both 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.

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, 23, 24]
[4] Easy integration into existing processes. [6, 22]
* Only partial support.

Luckily, the open-source scene of AI/ML/DS tools, libraries, frameworks, 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. 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 [26].

IBM's AutoAI [27] 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 primarily on non-technical users by providing them with a graphical user interface (GUI) 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 GUI for laypeople, and it has also been set to be retired by 2024. Its successor is Azure Machine Learning which shares many similarities with AWS's SageMaker suite [28].

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, 23, 29]. Among others, it promotes using 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 a 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) [30] and TorchX[4]. 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 [31]. There is no augmentation for most deployment best practices. Given the tight coupling between these libraries and their corresponding ML frameworks, they cannot generalise to models or algorithms of other frameworks and technologies. The Open Neural Network Exchange[5] format could be an option for overcoming these incompatibilities. However, wider support would be needed for seamless integration.

Open-source platforms also exist, such as MLflow and Seldon Core. They both rely on Kubernetes to provide their features. MLflow emphasises 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 [32]. 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, Ambassador/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 must be created to use Seldon's Custom Resource

[4]pytorch.org/torchx/latest

[5]onnx.ai

Definition. These are minor obstacles if the project is already built on top of K8s; however, even then, software engineers with solid cloud and DevOps backgrounds are actively required to use Seldon Core.

Additionally, increasing attention is given to ML deployments in embedded systems both from a theoretical [29] and practical [33] point of view. Prado et al. [33] survey the available deployment frameworks and end-to-end solutions, including those for embedded devices. They note the inefficiencies of these 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 to fit their changing needs and advancements in the field.

At the same time, Meenu et al. [29] present and 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 software engineering practices which have been ignored or had a very narrow focus in ML literature"*. In summary, the issues expressed in Section 2.2 can be understood when looking at the available solutions.

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 widespread adoption, leads us to believe that current frameworks are inadequate for many contexts, especially in cases where teams lack the background in cloud, operations, and more generally, software engineering. 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, Hugging Face, and similar libraries for the domain of AI/ML remains unmatched in the field of AI Engineering and MLOps.

Chapter 3

Methods

The chosen methodology for this study is Design Science which emphasises the need to design and investigate artifacts in their contexts [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 strategy seems fitting for our problem in consequence of its practical nature.

The design cycle shares similarities with Action Research [34] 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 [35], 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 for their weaknesses by applying multiple methods. Hence, the study also relies on interviews with professionals to validate the design decisions and determine the generalisability of *GreatAI*.

3.1 Design cycle

The aim of *GreatAI* can be summarised using the terminology of design science in the following way: *Facilitate the 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 of 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 which aims to find tech-transfer opportunities in academic publications. The primary input of the system as a whole is a set PDF files, while the output is a list of metrics describing various aspects of each paper, such as interesting sentences, scientific domains, and contributions. The result 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 (NLP) 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 within their organisation's core processes. Since the final ranking affects real people, explainability and robustness are also central questions.

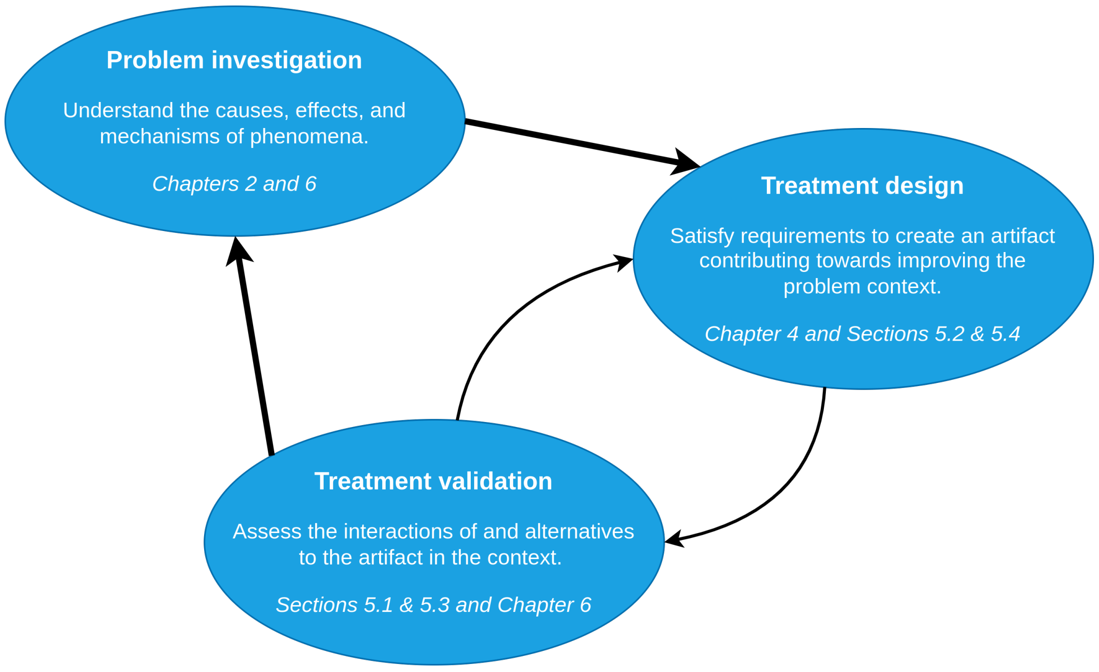

Problem investigation
Understand the causes, effects, and mechanisms of phenomena.
Chapters 2 and 6
Treatment design
Satisfy requirements to create an artifact contributing towards improving the problem context.
Chapter 4 and Sections 5.2 & 5.4
Treatment validation
Assess the interactions of and alternatives to the artifact in the context.
Sections 5.1 & 5.3 and Chapter 6

Figure 3.1: Implementation of the design cycle of design science [10] for our problem context of AI/ML deployments. The thinner arrows denote smaller but more frequent iterations.

The goal is to find a simpler, less cognitively-straining-to-use design that still leads to high-quality deployments, the definition of which will be described in Section 4.2. Before generalising, the framework's design is iteratively refined using the feedback acquired from applying it in practical contexts, which in this case are the research and development of a smaller and a more complex AI component using the work-in-progress framework.

The design cycle summarising the research approach is shown in Figure 3.1 indicating the role of the case studies. The concerns arisen in the *Treatment validation* iterations and their short discussions are highlighted in the form of *Design notes*. Afterwards, they are addressed in the following *Treatment design* iteration. This way, the issues are immediately considered and the proposed solutions can be traced back to the problems prompting their introduction.

3.2 Applicability & generalisability

To conclusively answer **RQ3** and **RQ4**, we conduct interviews with software engineers and data scientists with varying levels of professional background. The interview candidates were recruited from the recommendations of my acquaintances, who were kindly asked to seek out people from their professional networks with any connection to AI/ML. After the first few interviews, participants were also asked to suggest other candidates, preferably from different subfields. After two iterations of reaching out to potential interviewees personally, ten engineers and researchers eventually responded positively and participated in the study. Albeit the sample size is small, it still represents a wide range of organisation types: experts were included from startups, consultancies, government organisations, and research companies.

First, before their interview, participants 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*. 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, interviewees are asked to solve a real-world deployment task by finishing a partially completed example project[1] using *GreatAI*. This is a more straightforward instance of the AI development lifecycle presented in the *GreatAI* tutorials. They are also encouraged to think aloud so 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 a larger number of implemented best practices — can be quantitatively analysed by comparing the qualities of the finished implementation with the previously given answers. The target duration for the interviews is approximately one and a half hours.

We follow the guidelines proposed by Halcomb et al. [36] for collecting information from interviews and reporting it. This reflexive, iterative process starts by recording participants (with their permission) and concurrent note-taking. Reflective journaling is immediately done post-interview, which is subsequently extended and revised by listening to the recordings. Afterwards, we interpret the gathered information by applying the methodology of thematic analysis [37]. Thematic analysis is an iterative qualitative investigation technique consisting of labelling, correlating, and structuring the central recurring topics raised during discussions. It has been successfully used in previous software engineering studies for extracting emergent patterns [6, 38].

The second half of the one-on-one sessions consists of a short survey allowing us to create the Technology Acceptance Model (TAM) [39] 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 [40], and due to its general psychological origins, it proves to be effective in other areas of technology, not just software [41].

We employ the parsimonious version of TAM, which has been measured to have similar predictive power to that of the original TAM while having fewer variables [42]. 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 ten questions corresponding to these aspects of their experience using *GreatAI*. The questionnaire is shown in Appendix B. The internal consistency of the answers is calculated using Cronbach's Alpha [43], after which we reflect on the responses.

[1]Available at github.com/schmelczer/great-ai-interview-task. The training part of the task has already been done, and the participants only have to deploy the trained classifier.

Chapter 4

Designing the framework

Providing users with a high level of abstraction is not unheard of in the context of practical AI/ML platforms. Many software-as-a-service products offer features for hiding the technicalities of machine learning. However — as we discussed 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 saw in Chapter 2, these have limitations either from the perspective of best practices or stemming from their difficulty in being adopted. The scope has to be well-defined and limited to provide the best chance of providing an easy-to-adopt solution. To understand the API of a library, users first need to understand its aim and surface and have to become familiar with the problems it solves. Thus, limiting the focus solely to the *transition* step seems reasonable. This step is highlighted in Figure 4.1.

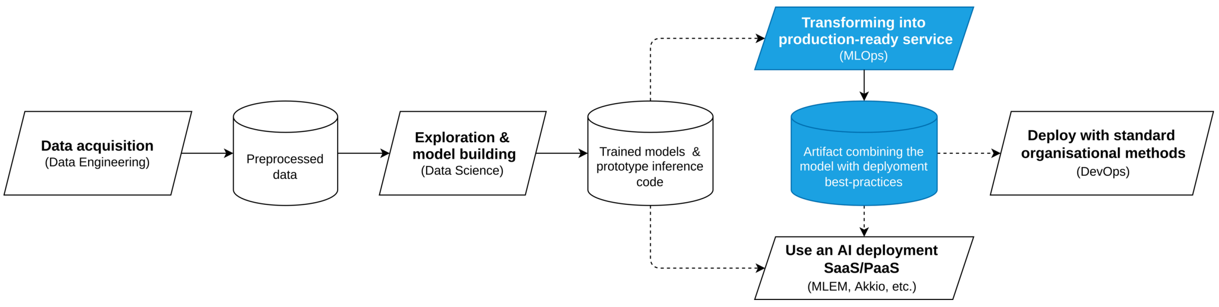

Data acquisition
(Data Engineering)
Preprocessed data
Exploration & model building
(Data Science)
Trained models & prototype inference code
Transforming into production-ready service
(MLOps)
Artifact combining the model with deplyment best-practices
Deploy with standard organisational methods
(DevOps)
Use an AI deployment SaaS/PaaS
(MLEM, Akkio, etc.)

Figure 4.1: Usual process steps (based on [24]) 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*.

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 best practices. Then, it is either deployed using a deployment SaaS or the organisation's existing software deployment setup.

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, 23] and John et al. [24]. The core requirements — set of covered best practices — for a software solution that has the potential to improve 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 frequently 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 four popular choices which offer varying but similar features: Alibi detect, Facets, Great Expectations, and Data Linter [44]. The responsibility of choosing the most fitting solution falls on the user. Thus, they should not be limited in this by *GreatAI*. On the contrary, the programming language (PL) of the library may be its only non-general property. Fortunately, the de facto PL for data science is Python, so implementing the library in it should not significantly limit its applicability.

Robustness In software development, robustness can be achieved by preparing the application to handle errors gracefully, even unexpected ones [45]. 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 safeguards as practicable. *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 system's real-world performance, which 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.

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

Trustworthy As detailed in the *Ethics guidelines for trustworthy AI*, 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.

The requirements were chosen stemming from their general importance and potential to be mostly 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 being solved, let us detail the principles we use while implementing their solutions. As implied in Section 4.1, the Unix philosophy [46, 47] of software design is followed. Most notably, the design goal that encourages to *write programs that do one thing and do it well.* 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* [48]: 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. In contrast, depth implies that each accomplishes 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 with a lot of functionality maximises return on investment (ROI), hence, developer experience (DX).

Moreover, the theoretical frameworks presented in *The Programmer's Brain* [49] 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 [50], and linguistic anti-patterns [51]. The former comes with a set of dimensions describing different (often competing) cognitive aspects of code that influence one's ability to perform specific tasks.

Linguistic anti-patterns 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, making it quicker to comprehend and easier to reason about the code [52]. Finding the most accurate and useful names is more challenging 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 [53]. In short, good names are essential to good APIs; consciously considering the implications of names must be an integral part of the design process.

Nonetheless, simple APIs come with a high technical cost. The library has to implement these in a way that still allows for high performance in production [54] and avoids being tied to specific libraries or technologies. Inspiration for the latter may be gained from the ML pipelines of Prado et al. [33]: they show that more freedom can be achieved with plug-and-play steps and preconfigured defaults.

4.3.1 Default configuration

Existing frameworks frequently suffer from the entanglement of numerous levels of abstractions.[2] Instead of exposing each implementation detail and encouraging users to interact with most of them, these can 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.

[2]grugbrain.dev/#grug-on-apis

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 [49]. Similar features can be imagined for providing a service API for the algorithms, giving feedback, marking outliers, and more.

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 handy WebGL library — TWGL.js, has a feature for automatically guessing the type of vectors based on their names. Suppose it matches the `/colou?r/i` pattern. In that case, it is treated as a vector with three components[3]. It is easy to imagine that this can help in certain scenarios. Still, 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 guessing must be avoided to create 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 vastly 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 [48]), 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. However, 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 skipping them.

This way, the library attempts to notify its user about the existence of these decisions but does not force them to decide manually. 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. Documentations come in many shapes: modern integrated development environments (IDEs) tend to show a popup of a function's description when requested (for instance, on mouse hover), but at the same time, a more comprehensive online manual and example projects are also still expected. Descriptive error messages can also be viewed as documentation.

The library must have quality documentation for all categories. Accordingly, for structuring it, the *Diátaxis* philosophy is preferred [55] 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, references, and explanations.

Once again, we might notice two competing interests: the level of detail and the length of the documentation. For example, FastAPI[4], a popular Python web framework, has extensive descriptions and explanations on all topics related to Python's import system, the HTTP protocol, concurrency, deployment, and more. The actual framework's documentation is sprinkled over these overly broad topics. This is undoubtedly helpful for beginners to acquire knowledge from a single place. Yet, this high level of

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

[4] fastapi.tiangolo.com

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 is 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, and 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 the documentation when they are stuck, and there might be some merit to this. Helping them not get stuck — by providing a starter code from which they can explore the API using IntelliSense-like solutions — should be preferred. Take the example of another widely popular Python web framework, Flask[5], at this time, has 324 homogeneously styled links on its landing page. Out of these, only two lead to the quick-start code. Of course, it is not hidden, but we 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 of good DX are consistency and discoverability. To give one more example, the MySQL connector's Python implementation[6] 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 is not always desirable; therefore, it is even more imperative to name similar functions similarly. This helps discoverability and chunking [49], which amounts to quicker comprehension.

There is one more reason to prefer consistency: humans have limited short-term memory (STM) [56]. Even though flags as function parameters are frowned upon by some [57], they can be 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 simultaneously, 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 never encounter this since the doubly-negated version is the default; thus, it is only singly negated when overriding it. This approach also implies that something may be recommended to be turned on by default.

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

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

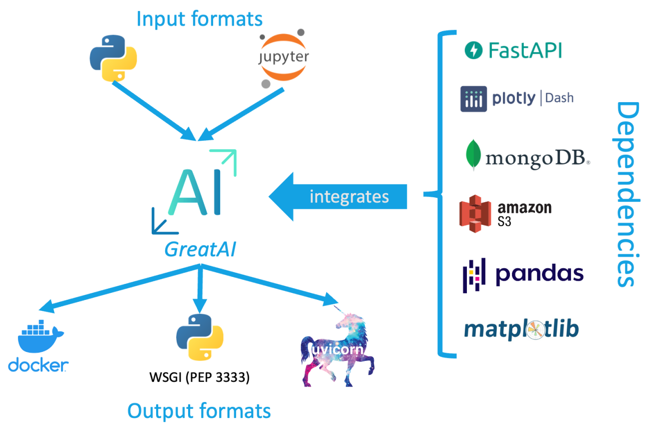


Figure 4.2: A very high-level overview of `GreatAI` in its context. The main dependencies are also highlighted.

4.4 Architecture

Although API design has been the central subject so far, it is worth remembering that APIs are usually expected to have corresponding implementations. *GreatAI* is no exception. As laid out in Section 4.3, we strive for narrow and deep interfaces; thus, it is time to address the *depth* component.

GreatAI stands on the shoulders of numerous open-source packages and integrates them to provide its various features. The most fundamental dependencies and the entire library in context are shown in Figure 4.2. Given a Python script or a Jupyter notebook, *GreatAI* transforms the specified prediction functions into a production-ready deployment, deployable either as a Docker image, WSGI-server, or an executable relying on `uvicorn`. The complete list of dependencies can be found in the repository[7].

The general theme in the implementation is that each explicit best practice should have its distinct, loosely-coupled functions or classes. When collaboration opportunities arise, such as persisting the model versions (1st component) into prediction traces (2nd component), there are three primary conduits for realising them. These are the `context` object responsible for the global configuration per process, the `FunctionMetadataStore` specifying the expected behaviour of each prediction function, and finally the `TracingContext` that is created anew for each prediction input (session).

After refining the framework with feedback gathered from case studies and users, we will end up with the core architecture presented in Figure 4.3. The implementation is mixed-paradigm, combining the expressiveness of functional and the design patterns of object-oriented programming (OOP) in order to maintain an overall low complexity. Reflection is also utilised, especially for run-time type-checking and generating the API definitions and dashboard components. Regardless, the architecture is still presented with a syntax similar to the class diagrams of UML2 [58] because it provides the freedom to express even the non-OOP design aspects.

For the sake of brevity, Figure 4.3 does not show all fields, and some related entities have been combined, e.g. the *GroundTruthAPI* box represents the `add_ground_truth`, `query_ground_truth`, and `delete_ground_truth` functions. The client project can also access most of the presented entities, but these optional dependency arrows are not shown in the diagram. The `utilities` submodule is also left unexpanded; almost all of its functions are orthogonal with the exception of `parallel_map`. The latter follows a textbook producer-consumer model facilitated by queues and event signals [59].

[7]github.com/schmelczer/great-ai/blob/main/pyproject.toml

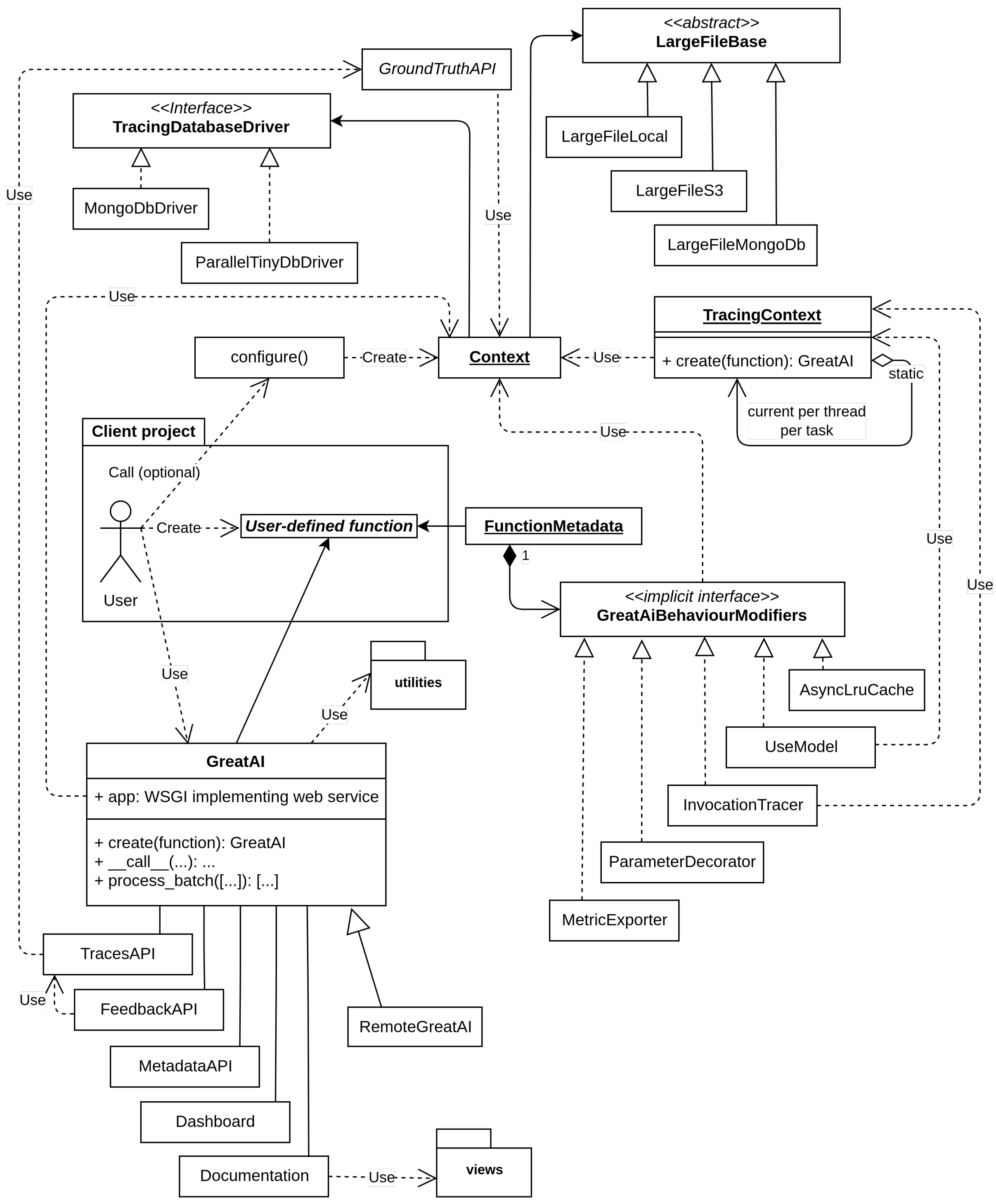


Figure 4.3: The core architecture of *GreatAI* illustrated with syntax loosely-based on UML2 [58]. Given its framework nature, the expected client project and the actor integrating it are highlighted; the associations between the framework and the client project are achieved through the use of decorators.

Chapter 5

The ScoutinScience Platform

The core product of ScoutinScience B.V.[1] is its Platform[2]. The clients are Technology Transfer Offices of Dutch and German universities, government organisations (e.g. Wetsus), and corporations (e.g. Heraeus Group and 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 markers 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 [54] 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, and it is also larger than it should be. This has 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 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 that can hopefully advance the state-of-the-art.

This chapter describes the process of designing *GreatAI* and how it fits into real-life use cases. First, a simple experiment is presented which investigates a Naïve Bayes classifier's [60] accuracy at predicting the fields of papers. This leads to the implementation of a software service that is deployed to production. Subsequently, as the feature set of the library grows and matures, a more complex component is developed concerning text-summarisation with SciBERT [61]. After implementing each case, the insights gained are fed back into the library's design.

[1] scoutinscience.com

[2] dashboard.scoutinscience.com

5.1 Domain classification with Naïve Bayes

Using different models for slight variations of the same problem is 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 [25]. We have also found that in order to best process an academic publication, knowing its domain is essential. One of the reasons for this can be 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). In contrast, 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 classifier 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 [60] for precisely 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. [62]. They created a 176-class classification problem for comparing bibliometric and deep-learning approaches. However, this comparison is made difficult because 44% of the labels are *assigned suboptimally* in the ground truth dataset.

Prior work evaluated SciBERT [61] — a BERT [16] model pretrained on academic publications — on a simpler version of the task in which the domains of sentences[3] have to be decided[4]. It achieved an F1-score of 0.6571 after being pretrained on the Semantic Scholar Corpus (SSC) [64] and fine-tuned on the train split of the Microsoft Academic Graph (MAG) dataset [65][5]. To our knowledge, no other published work exists on this sentence classification task. This may be explained by the task's lack of practical relevance and contrived nature (uniform label distribution), as we will see in the following 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 are 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 to MAG's 7, and it also contains abstracts instead of just sentences; thus, it is more fitting for our practical use case.

SciBERT's version of the MAG dataset has 84,000 and 22,300 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.

[3]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 [63].

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

[5]SciBERT was applied to a preprocessed version of this dataset, available at: github.com/allenai/scibert/tree/master/data/text_classification/mag

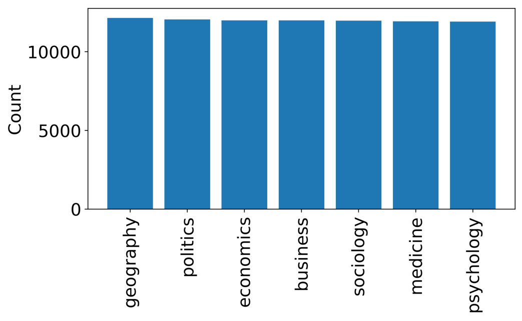


Figure 5.1: Class distribution of the MAG [65] dataset's 84000 sentences in its *train* split.

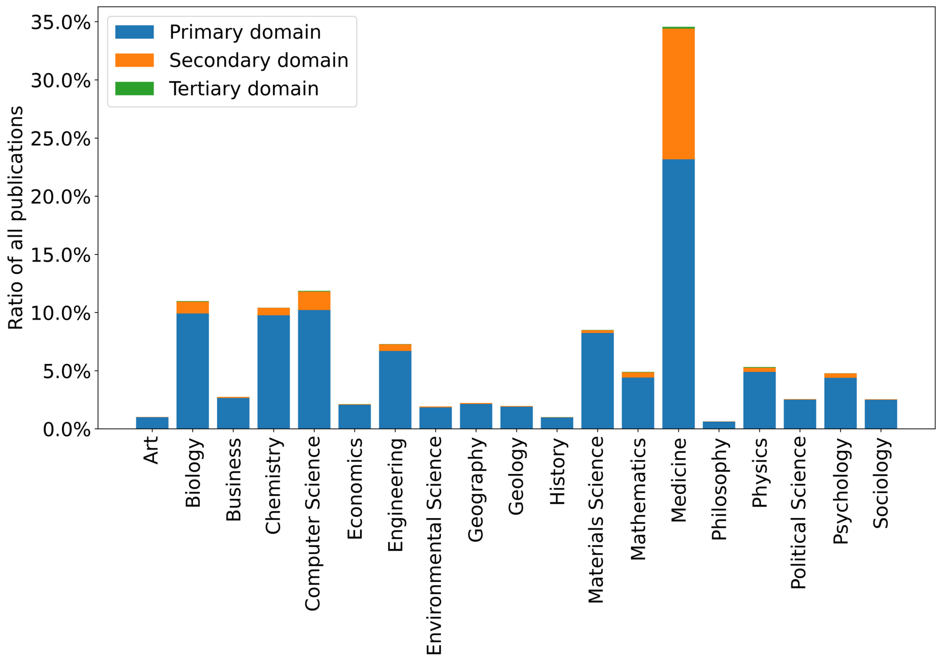


Figure 5.2: Label distribution of the Semantic Scholar dataset [64]. Each publication may be assigned at most three labels.

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 are randomly downsampled to give us 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 also be 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 [61]. However, 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 normally 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 side-effects.

5.1.3 Methods

Our aims are twofold: (1) to evaluate a sentence classification model on MAG and compare it with the 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. For testing this hypothesis, a unigram language model — Multinomial Naïve Bayes (MNB) — 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 [19], it only took 71 lines of code to create, hyperparameter optimise, and test a text classifier.[6] This further proves 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) [66], the hyperparameters found via 10-fold cross-validation on the *train* split lead to filtering out tokens which occur in fewer than five documents or more than 5% of the documents.

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, there seems to be 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, which is 2.3% more than SciBERT's on the same dataset. Thus, it seems that MNB clearly outperforms SciBERT for this particular use case: it is not only more accurate, but its model is magnitudes smaller. At the same time, it is also considerably faster to train (or fine-tune 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 choosing MNB for the task over SciBERT.

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

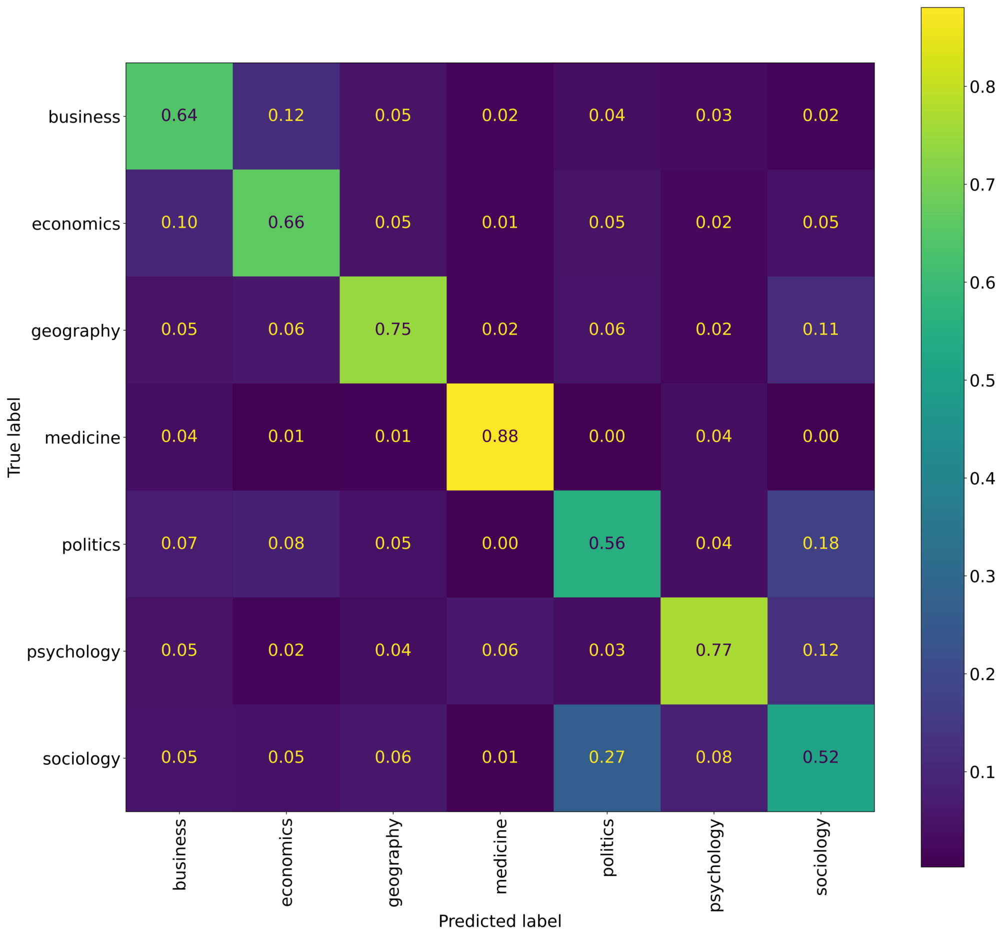


Figure 5.3: Confusion matrix of a Naïve Bayes classifier on the MAG dataset's sentences. The matrix is normalised column-wise. Notice, how most mistakes happen between semantically similar classes, for instance: *politics – sociology* or *business – economics*.

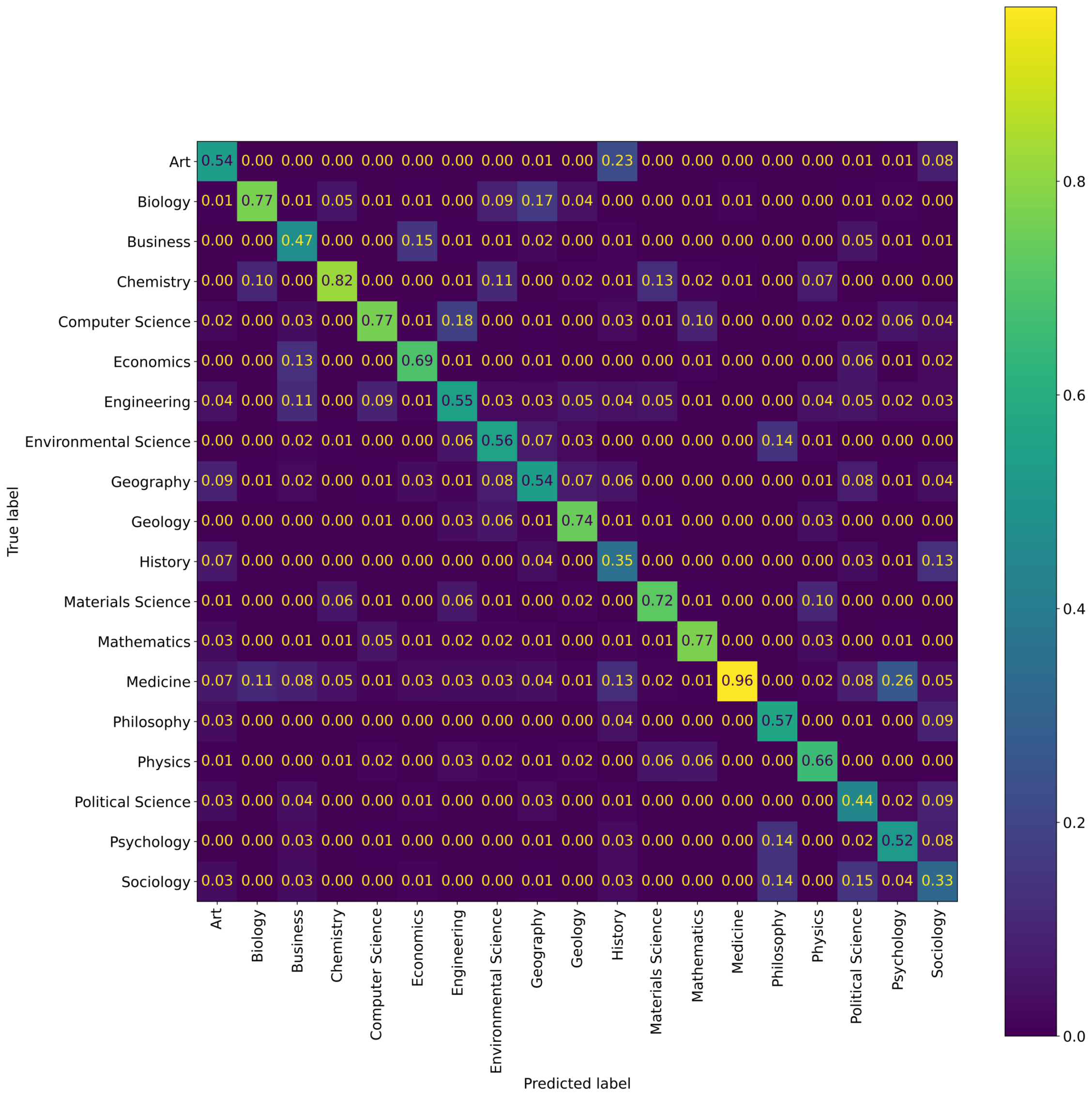


Figure 5.4: Confusion matrix of a Naïve Bayes classifier on the SSC dataset's sentences. The matrix is normalised column-wise. Notice, how most mistakes happen between semantically similar classes, for instance: *philosohpy – sociology* or *history – art*.

It is, of course, not entirely surprising that the sophisticated transformer architecture of SciBERT is not necessary for a straightforward task like this. Apart from phrases, the relations between separate words of a sentence do not carry nearly as much discriminative power as the identity of the terms [67]; 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 a lower macro-average than on MAG, with an F1-score of 0.59.[7] The weighted-average F1 is 0.70, and the overall accuracy is also 70%. The substantial difference between the macro and weighted averages comes from the unbalanced distribution of the labels. The lower F1-score is not surprising because this dataset has more than twice as many classes. Additionally, the mistakes made are defensible when we look at Figure 5.4: most of them are between related domains.

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 to take input on the fly and calculate a corresponding prediction. Since we aim to follow the best practices *Explain Results and Decisions to Users* and *Employ Interpretable Models When Possible*, explaining 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 five tokens from the input text ranked by their feature weights. The last deployment step is 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 improve its consumers' DX. 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, and sometimes, they even follow the conventions of REST. Either way, we must develop a wrapper over the service to make it available to 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.

[7]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 prediction 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

Let us first 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 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, we can conclude that assuring its correctness lies within the scope of *GreatAI*. Henceforth, cross-cutting concerns should be covered.

This section briefly explores how the problems raised can be solved using *GreatAI* and the API it provides 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 are 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 [68] 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.

We must address two data storage needs: training data and trained models. Proper version control is one of the most basic expectations for commercial codebases. Based on developer surveys, it is likely that our code is already tracked under Git and synchronised with GitHub[8]. Therefore, using Git Large File Storage (LFS) 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[9].

An open-source tool, the Data Version Control (DVC)[10] provides a nearly perfect alternative. It comes with a command-line interface (CLI) inspired by Git's and 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 we highly recommend opting for DVC. Nevertheless, if this may prohibit a team[11] from properly handling data according to the best practices, we present a simpler solution.

[8]octoverse.github.com/#lets-look-back-at-the-code-and-communities-built-on-git-hub-this-year

[9]docs.github.com/en/repositories/working-with-files/managing-large-files/removing-files-from-git-large-file-storage

[10]dvc.org

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

The complexity of an API can be decreased by relying on its users' preexisting knowledge, and known patterns [49, 48]. Therefore, we can reuse familiar APIs, such as the `open()` method from Python. 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[12]: the same parameters can be used with it resulting in similar observed behaviour. The expected features: versioning, progress bars, caching, garbage collecting the cache, and automatically deleting old remote versions are all present and come with recommended — but easy to see and change — configuration.

Easing development is not merely about automating everything but also about 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 also be given to lock ourselves in using a specific version which can be desirable 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 are some of the most common. But splitting sentences, language tagging, and robustly lemmatising 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 could be extensively tested through 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, we 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 also implemented which closely mimics the API of the built-in Python function: `map`. Furthermore, it exemplifies how even a close to trivial function can improve the DX by magnitudes. Rooted in the global interpreter lock (GIL)[13] 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 standard `multiprocessing` library has a great API. However, doing a parallel mapping task with a progress bar still takes about a dozen lines. 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 one-liner 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 already been solved by software engineering, more specifically, by the DevOps paradigm [69]. In line with the findings of John et al. [24] on the SOTA of AI deployments, we

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

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

suggest wrapping the applications in a format 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 experiences, three types of software artifacts are identified (in the context of Python) for which a wide range of established practices exist. WSGI server[14] compatible applications, executable scripts, and Docker Images[15]. To achieve this, *GreatAI* provides a compatibility layer between simple Python inference functions and all the abovementioned common artifacts. Taking functions as input for the first step also satisfies the requirement to be **General**. Nevertheless, to also allow customisation, additional configuration, metadata, and behavioural specification can be given as well.

```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` could be the inference function of an ML model.

The main advantage of the wrapping approach is that it does not require any input from the clients (by default). We opted for a decorator [70], 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 save_model, GreatAI, parameter, use_model, log_metric

# this could have been called in another script
save_model('special_number', 405)

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

assert add_number(12).output == 417
```

Listing 2: A simple *GreatAI* service with behavioural customisations.

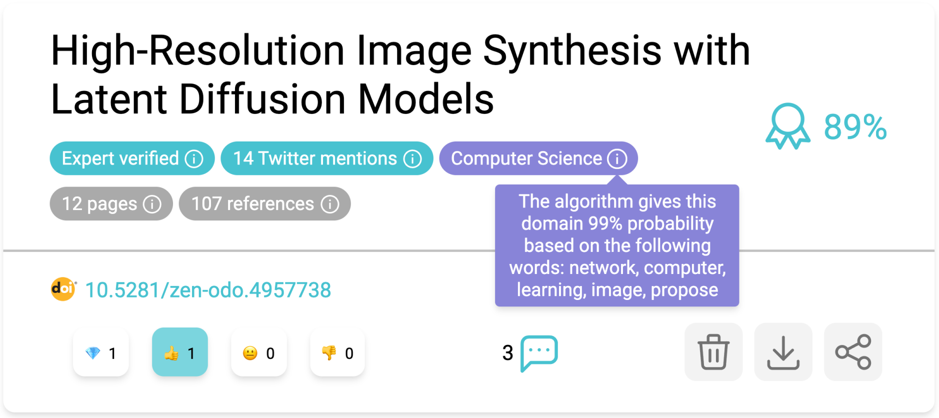
[14] peps.python.org/pep-3333

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

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 essential 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 those with the inputs producing them 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 library functions. An example of this can be seen in Listing 2, while more details of the semantics can be found in the documentation[16].

5.2.4 Summary


High-Resolution Image Synthesis with Latent Diffusion Models

Expert verified · 14 Twitter mentions · Computer Science · 12 pages · 107 references · 89%

The algorithm gives this domain 99% probability based on the following words: network, computer, learning, image, propose

10.5281/zen-odo.4957738

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

After implementing some features of the library, it can already be 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 into a Docker image. This image can be instantiated by the company's existing DevOps pipeline and cloud infrastructure. In 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 the framework fares in a more complex case.

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

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 with the help of a word2vec model [71]. The user feedback deemed this implementation slightly helpful but inadequate for providing an accurate overview. Thus, this is the baseline we 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 [72]. 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 [73]. 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 [74, 75], 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 on the granularity of sentences. This gives rise to an obvious idea: show the experts something they can annotate. Because experts' time 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) [76, 77, 78, 79] provides a promising direction to do so.

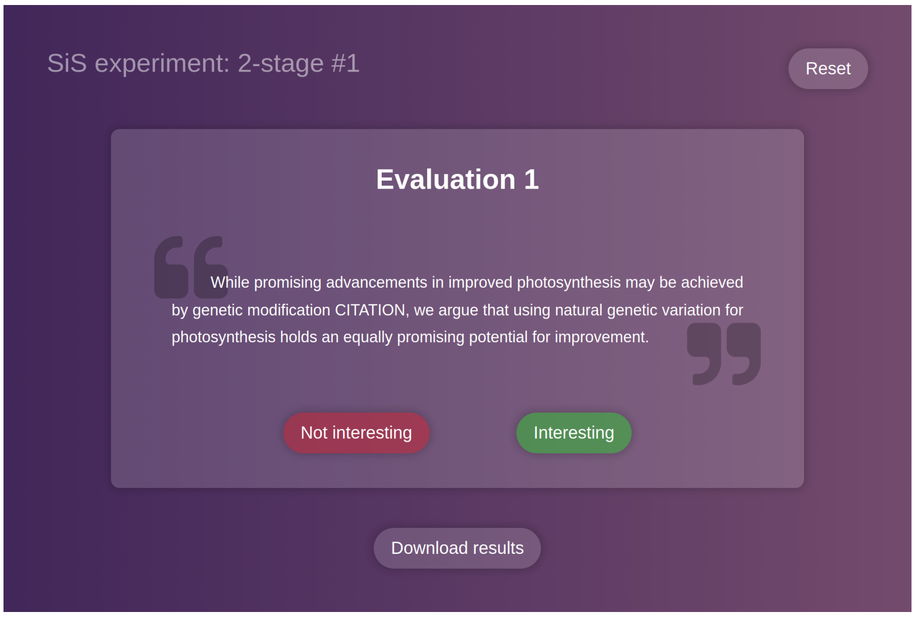
A formulaic expression is a phrase with zero or more "slots" which, when filled appropriately, leads to expressing a certain intent. In the context of scientific text, an example[17] could be: `it was not until * that`. The asterisk can be substituted with multiple terms, and the intention of this expression is (likely) to describe the *History of the related topics*. Iwatsuki et al. identified a set of 39 intentions, compiled a manually labelled dataset [76], and developed multiple approaches for automatically extracting and classifying formulaic expressions in large corpora [78, 79].

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 information retrieval [80]. The filtering is expected to sieve

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

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. Figure 5.6 shows an example of the annotation task. Our method turns the extractive summarisation into a binary classification task for which a SciBERT model [61] can be fine-tuned. Ultimately, the summaries are derived from sentences selected by the classifier trained on the experts' annotations.



While promising advancements in improved photosynthesis may be achieved by genetic modification CITATION, we argue that using natural genetic variation for photosynthesis holds an equally promising potential for improvement.

Figure 5.6: The annotator GUI showing a single sentence and the two labels that can be assigned based on its relevance to technology-transfer.

We have to note two possible shortcomings of this setup: firstly, the FE intentions are assumed to be strongly correlated with the sought-after *tech-transfer opportunities* 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. Nonetheless, we expect that stemming from the length of the documents, and the sparseness of the selected sentences, any combination of them is likely to have low redundancy.

5.3.3 Results

For the first iteration, 1500 sentences were selected for two experts to annotate in a binary fashion according to strict guidelines. Afterwards, for measuring the interrater agreement, we calculated Cohen's kappa [81] as shown in Equation 5.1, which turned out to be **0.43** for the two annotators. This happens to be just above the lower end of *moderate agreement*. Even though the original quality ranges are sometimes criticised for being too relaxed for the medical domain [82], some leniency is acceptable for many NLP tasks due to their subjectiveness. Regardless, in the case of summarisation, Verberne et al. [83] 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. This is reasonable because the annotators have dissimilar backgrounds and likely judged slightly different aspects of the sentences.

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

Reproducibility Reproducible experiments are generally preferred. It is easy to forget to set some seed values and, for example, end up with different data points in the test-train splits during training and validation in a Continuous Integration (CI) pipeline, thus, data leakage. For facilitating 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 [84] on its own; however, in some cases, it can result in one fewer step to miss.

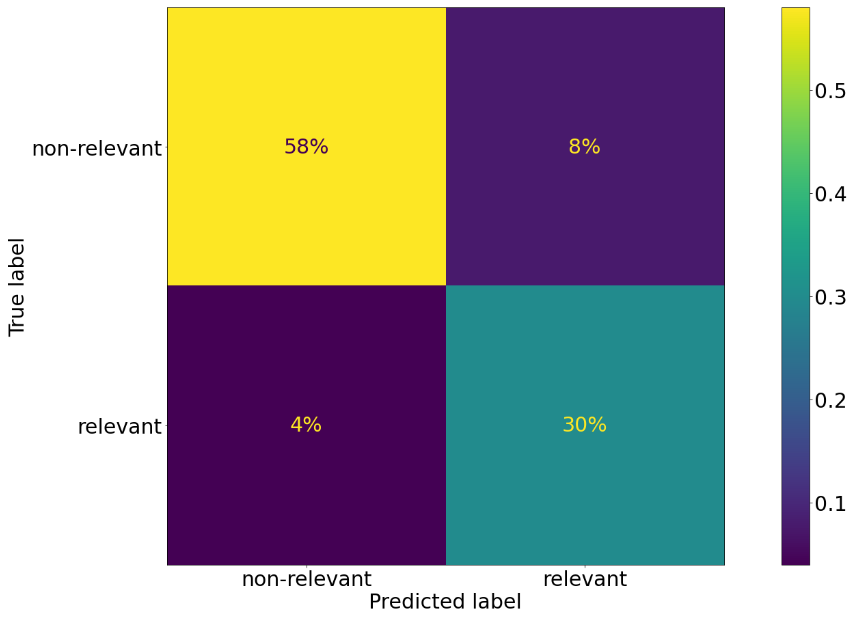


Figure 5.7: Confusion matrix of the fine-tuned SciBERT model on the *summary candidate sentences* dataset.

The next step is fine-tuning SciBERT with the help of Hugging Face `transformers` [15]. The data are divided into training and test splits with a ratio of 4:1. A validation split, used for early stopping, is also derived from the train split. The objective function is the F1-score of the positive class, and the early stopping patience is five epochs. The learning rate is 5×10^{-5} and AdamW [85] is used for optimisation with a weight decay of 0.05. The code can be found in the documentation[18], it is surprisingly slightly shorter than the code of Section 5.1.

Utility of LargeFiles For the purposes of the documentation, the fine-tuning 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 instance is granted. The `LargeFile` implementation alleviated this problem by automatically handling the uploads and downloads. Of course, first, backwards compatibility had to be solved for Python 3.7, the only available environment in Colab.

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

Table 5.1: Accuracty metrics of the fine-tuned SciBERT model on the *summary candidate sentences* dataset.

	Precision	**Recall**	**Support**
NON-RELEVANT	0.93	0.83	191
RELEVANT	0.73	0.88	109

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 relevant (summary candidate) sentences are supposed to reflect the same aspect of papers, we can reasonably expect some correlation between the grades and relevant sentence counts.

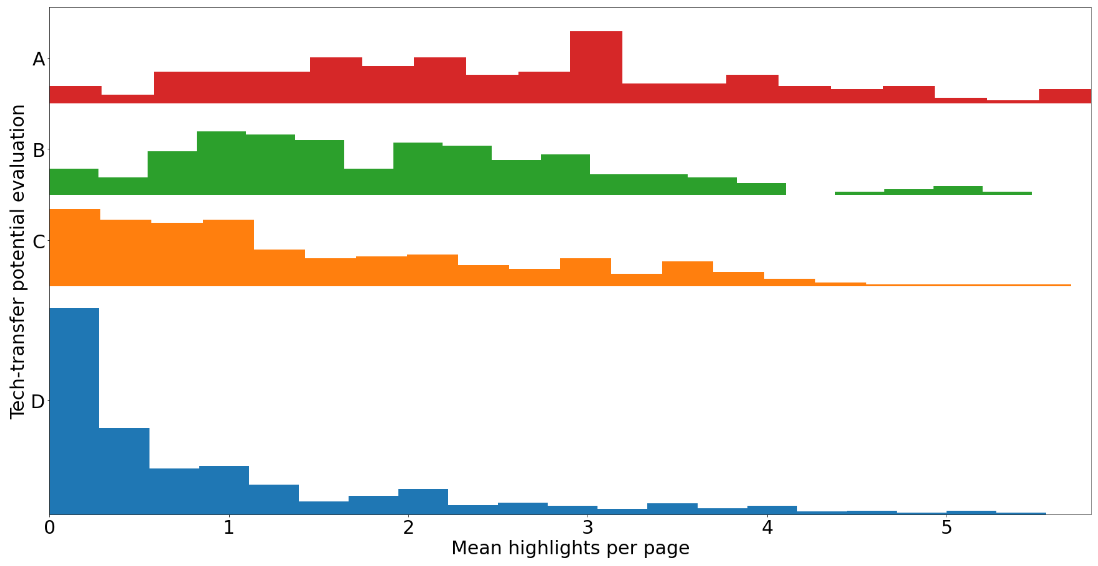
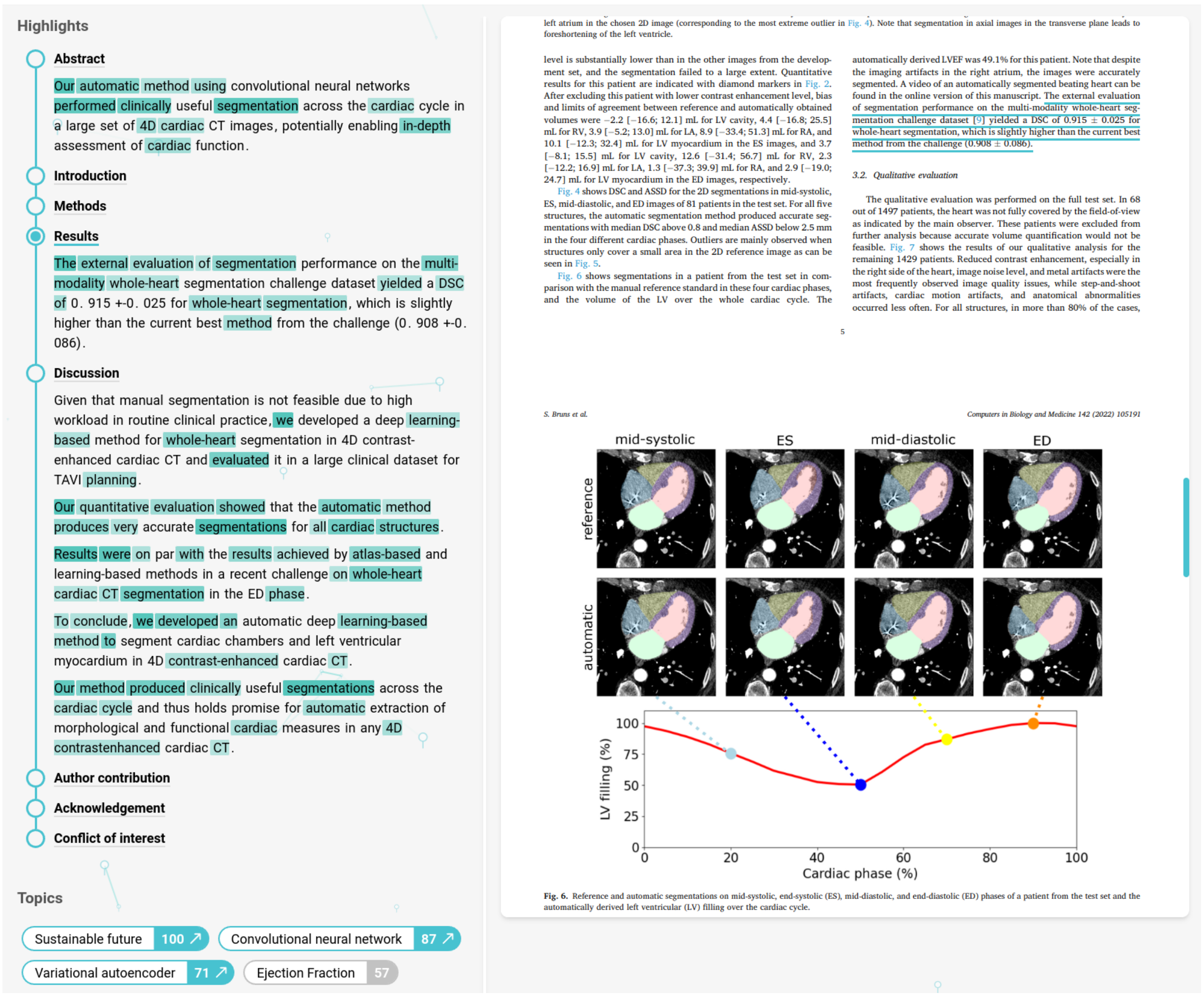


Figure 5.8: Distribution of mean predicted summary candidate sentence counts (refered to as *highlights*) in 4 categories. Category `A` corresponds to the most, while `D` to the least interesting papers based on mean user feedback. The sample size is 1406 (`D`=715, `C`=309, `B`=198, `A`=184). The histograms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

The best validation results were achieved after eight epochs which is 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, and the per class accuracy metrics in Table 5.1. Despite the task's subjective definition, SciBERT achieves good quality, indicated by an F1-score of **0.80**.

Figure 5.8 shows the ratio of summary candidate sentences as predicted by the fine-tuned model in 4 categories (grades) of papers. This dataset does not overlap with the training data; hence, the results come solely from the model's ability to generalise. It is interesting to see that the Spearman's rank correlation coefficient [86] between the normalised "highlights" counts and the ratings of papers 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 chosen by the


Highlights

Abstract

Our automatic method using convolutional neural networks performed clinically useful segmentation across the cardiac cycle in a large set of 4D cardiac CT images, potentially enabling in-depth assessment of cardiac function.

Introduction

Methods

Results

The external evaluation of segmentation performance on the multi-modality whole-heart segmentation challenge dataset yielded a DSC of 0. 915 +-0. 025 for whole-heart segmentation, which is slightly higher than the current best method from the challenge (0. 908 +-0. 086).

Discussion

Given that manual segmentation is not feasible due to high workload in routine clinical practice, we developed a deep learning-based method for whole-heart segmentation in 4D contrast-enhanced cardiac CT and evaluated it in a large clinical dataset for TAVI planning.

Our quantitative evaluation showed that the automatic method produces very accurate segmentations for all cardiac structures.

Results were on par with the results achieved by atlas-based and learning-based methods in a recent challenge on whole-heart cardiac CT segmentation in the ED phase.

To conclude, we developed an automatic deep learning-based method to segment cardiac chambers and left ventricular myocardium in 4D contrast-enhanced cardiac CT.

Our method produced clinically useful segmentations across the cardiac cycle and thus holds promise for automatic extraction of morphological and functional cardiac measures in any 4D contrastenhanced cardiac CT.

Author contribution

Acknowledgement

Conflict of interest

Topics

Sustainable future 100 ↗ | Convolutional neural network 87 ↗ | Variational autoencoder 71 ↗ | Ejection Fraction 57

left atrium in the chosen 2D image (corresponding to the most extreme outlier in Fig. 4). Note that segmentation in axial images in the transverse plane leads to foreshortening of the left ventricle.

level is substantially lower than in the other images from the development set, and the segmentation failed to a large extent. Quantitative results for this patient are indicated with diamond markers in Fig. 2. After excluding this patient with lower contrast enhancement level, bias and limits of agreement between reference and automatically obtained volumes were −2.2 [−16.6; 12.1] mL for LV cavity, 4.4 [−16.8; 25.5] mL for RV, 3.9 [−5.2; 13.0] mL for LA, 8.9 [−33.4; 51.3] mL for RA, and 10.1 [−12.3; 32.4] mL for LV myocardium in the ES images, and 3.7 [−8.1; 15.5] mL for LV cavity, 12.6 [−31.4; 56.7] mL for RV, 2.3 [−12.2; 16.9] mL for LA, 1.3 [−37.3; 39.9] mL for RA, and 2.9 [−19.0; 24.7] mL for LV myocardium in the ED images, respectively.

Fig. 4 shows DSC and ASSD for the 2D segmentations in mid-systolic, ES, mid-diastolic, and ED images of 81 patients in the test set. For all five structures, the automatic segmentation method produced accurate segmentations with median DSC above 0.8 and median ASSD below 2.5 mm in the four different cardiac phases. Outliers are mainly observed when structures only cover a small area in the 2D reference image as can be seen in Fig. 5.

Fig. 6 shows segmentations in a patient from the test set in comparison with the manual reference standard in these four cardiac phases, and the volume of the LV over the whole cardiac cycle. The automatically derived LVEF was 49.1% for this patient. Note that despite the imaging artifacts in the right atrium, the images were accurately segmented. A video of an automatically segmented beating heart can be found in the online version of this manuscript. The external evaluation of segmentation performance on the multi-modality whole-heart segmentation challenge dataset [9] yielded a DSC of 0.915 ± 0.025 for whole-heart segmentation, which is slightly higher than the current best method from the challenge (0.908 ± 0.086).

3.2. Qualitative evaluation

The qualitative evaluation was performed on the full test set. In 68 out of 1497 patients, the heart was not fully covered by the field-of-view as indicated by the main observer. These patients were excluded from further analysis because accurate volume quantification would not be feasible. Fig. 7 shows the results of our qualitative analysis for the remaining 1429 patients. Reduced contrast enhancement, especially in the right side of the heart, image noise level, and metal artifacts were the most frequently observed image quality issues, while step-and-shoot artifacts, cardiac motion artifacts, and anatomical abnormalities occurred less often. For all structures, in more than 80% of the cases,

5

S. Bruns et al. Computers in Biology and Medicine 142 (2022) 105191

Fig. 6. Reference and automatic segmentations on mid-systolic, end-systolic (ES), mid-diastolic, and end-diastolic (ED) phases of a patient from the test set and the automatically derived left ventricular (LV) filling over the cardiac cycle.

Figure 5.9: The tech-transfer summary of an academic publication ([87]). The titles and sentences can be clicked to navigate the paper on the right. Meanwhile, some explanation is provided by the highlighted words, the opacity of which corresponds to their attention weights.

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 are applied to the scores to avoid highlighting 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, Hugging Face 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 *Discoveribility*. For example, the attention weights for the GUI 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 our library design; it has to have an explicit and discoverable API that can be achieved using type hints, descriptive property names, and docstrings.

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 that were motivated by the *Highlights (summarisation) service* are presented. After which, the final surface of the API is described, which will be evaluated by its relation to the deployment best practices [4, 23, 24, 29] in the next chapter.

5.4.1 Caching

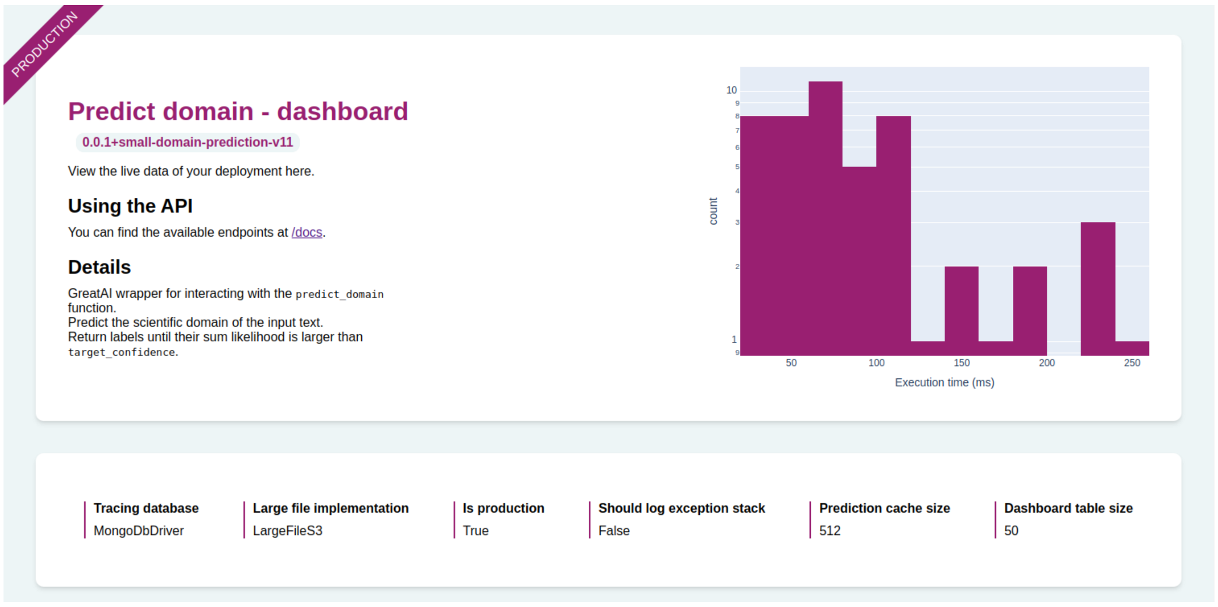
Sustainability is an increasingly crucial concern of ethical AI [88]. Without discussing the pros and cons of the green computing movement [89], we can still agree that computing time should not be wasted. To this end, caching the results of expensive operations has to be considered in any AI deployment. 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 (or GPU-bound), it turns out that sometimes they involve IO, especially when relying on the results of other remote models. This means a significant performance improvement can be achieved by implementing some inference functions asynchronously [90]. Thus, *GreatAI* also has to support decorating both regular (synchronous) and asynchronous functions. One notable consequence is that the batch processing feature must also be compatible with `async` inference functions. Batch processing is still a helpful 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` [91], and, another popular library, `joblib`[19] all lack the support for efficiently parallelising async functions. For this reason, `parallel_map` was 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.

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



Predict domain - dashboard

0.0.1+small-domain-prediction-v11

View the live data of your deployment here.

Using the API

You can find the available endpoints at /docs.

Details

GreatAI wrapper for interacting with the `predict_domain` function.
Predict the scientific domain of the input text.
Return labels until their sum likelihood is larger than `target_confidence`.

Tracing database	Large file implementation	Is production	Should log exception stack	Prediction cache size	Dashboard table size
MongoDbDriver	LargeFileS3	True	False	512	50

Figure 5.10: The header of the automatically generated dashboard of the service from Section 5.1. A generated documentation is shown on the left, while the histogram of response times is rendered on the right. The current configuration is prominently displayed on the bottom.

5.4.3 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. To ease the introduction of *GreatAI* services, a rudimentary dashboard is — optionally — generated. The dashboard's main features can be observed in Figures 5.10, 5.11, and 5.12. The diagrams and filterable/sortable table are interconnected and are automatically updated; the reactive behaviour is provided by the Dash framework [92].

5.4.4 Programmatic integration

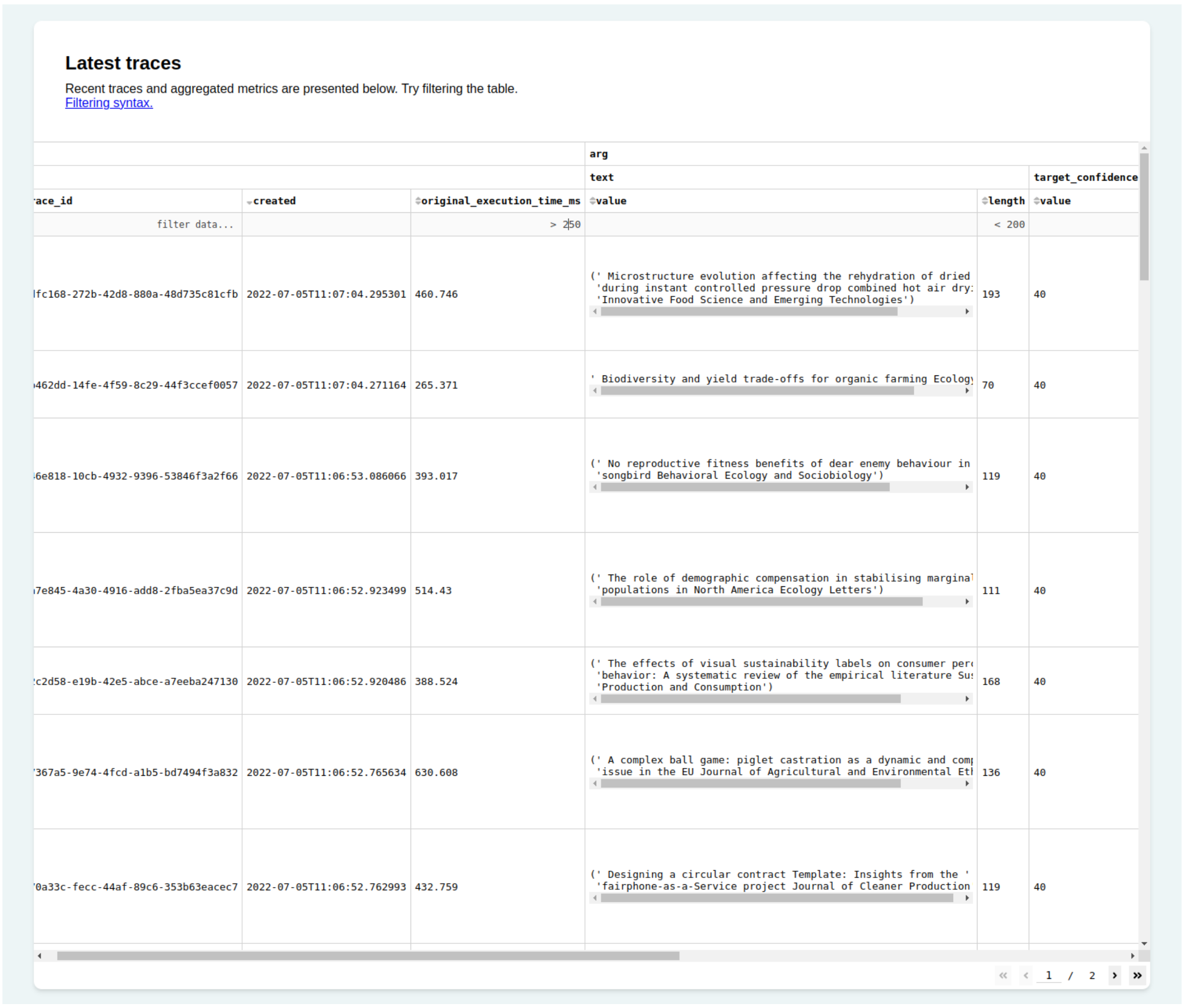
Apart from supporting `async` calls, a couple more steps can be taken to help integrate any service with a *GreatAI* deployment. This is implemented by the `call_remote_great_ai` function which hides the networking required to call a *GreatAI* instance's REST API. It takes care of validation, automatic retries, serialisation, and deserialisation. It 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 [93].

Additionally, a REST API is generated with its accompanying OpenAPI schema[21] 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[22] compatible. These can be seen in Figure 5.13. This, combined with a `/version` HTTP endpoint for programmatic access and validation of the service's metadata, proved to be valuable features when integrating the *Highlights service* into ScoutinScience's service-based architecture.

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

[21]swagger.io/specification

[22]semver.org



Latest traces

Recent traces and aggregated metrics are presented below. Try filtering the table.
Filtering syntax.

			arg		
			text		target_confidence
race_id	created	original_execution_time_ms	value	length	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 Ecology	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 perc 'behavior: A systematic review of the empirical literature Sus '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 comp 'issue in the EU Journal of Agricultural and Environmental Eti	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

« ‹ 1 / 2 › »

Figure 5.11: 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.

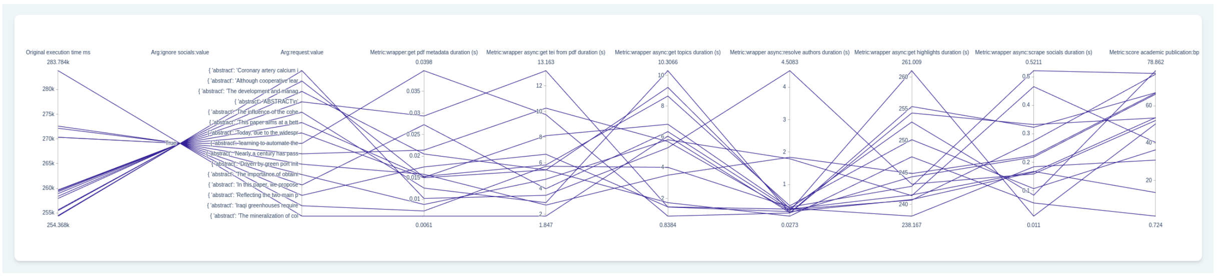
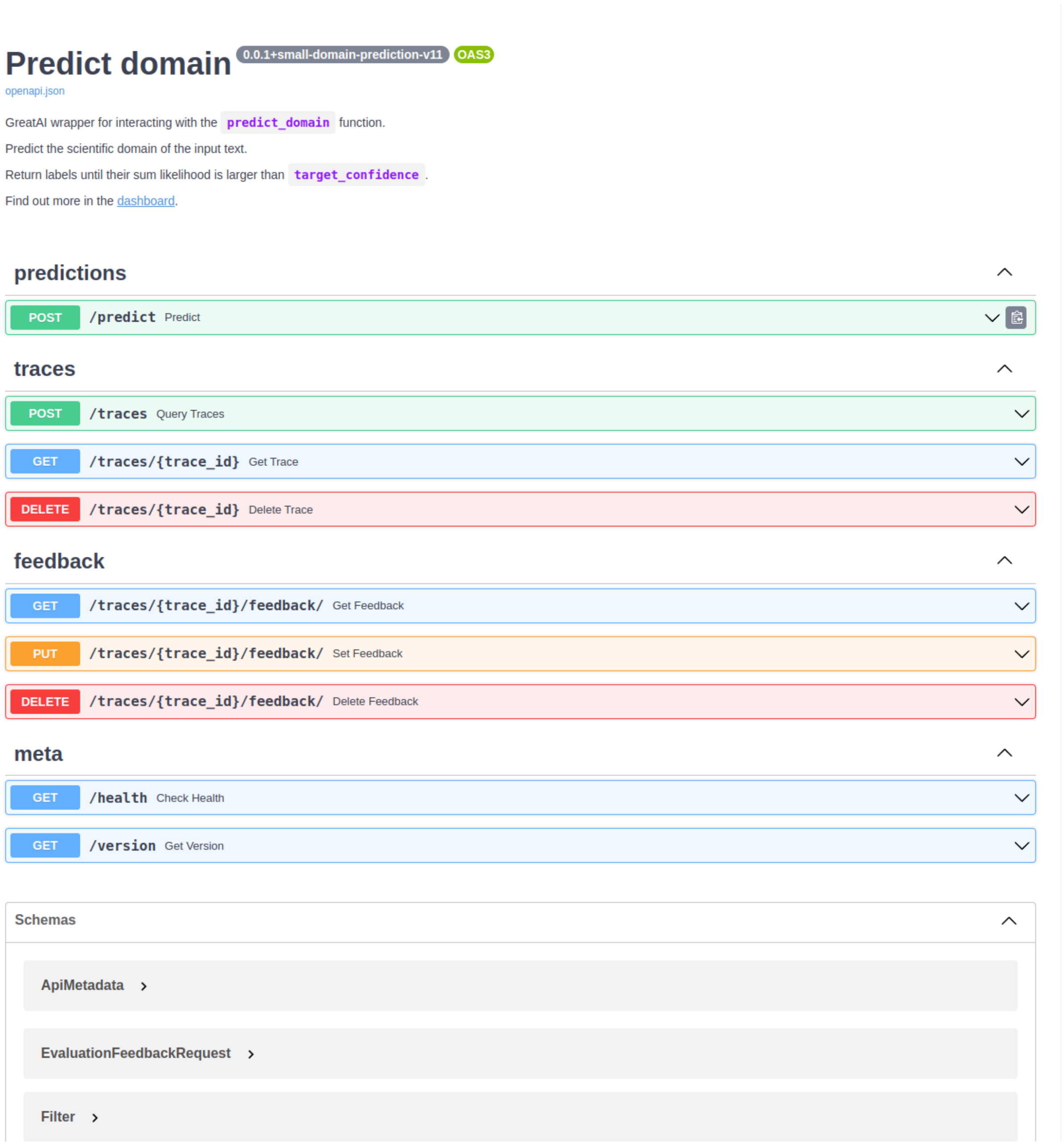


Figure 5.12: A parallel coordinates view of the traces displayed in the table above. Adding new axes is as easy as calling `log_metric` inside the inference function.



Predict domain 0.0.1+small-domain-prediction-v11 OAS3

openapi.json

GreatAI wrapper for interacting with the `predict_domain` function.

Predict the scientific domain of the input text.

Return labels until their sum likelihood is larger than `target_confidence`.

Find out more in the dashboard.

predictions

POST /predict Predict

traces

POST /traces Query Traces

GET /traces/{trace_id} Get Trace

DELETE /traces/{trace_id} Delete Trace

feedback

GET /traces/{trace_id}/feedback/ Get Feedback

PUT /traces/{trace_id}/feedback/ Set Feedback

DELETE /traces/{trace_id}/feedback/ Delete Feedback

meta

GET /health Check Health

GET /version Get Version

Schemas

ApiMetadata

EvaluationFeedbackRequest

Filter

Figure 5.13: 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`.

Chapter 6

Results & discussion

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. The objective was to design a library with an API that finds the balance between being simple enough to adopt without friction yet useful enough to be adopted. Simplicity is subjective and will be discussed separately in Section 6.2. For now, let us look at the utility of *GreatAI*.

6.1 Features

For answering **RQ3** — *To what extent can GreatAI automatically implement AI deployment best practices?* — a comparison is presented in the following, demonstrating a subset of best practices that 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 methods found by prior surveys of scientific and grey literature [4, 23, 24].

In order to show an accurately nuanced representation, a *Level of support* is determined for each best practice on a scale of *Partially supported*, *Supported*, and *Fully automated.* 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, various 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 type hints in their code and it contains a more complex control flow, it can only be partially type-checked. 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 an effort to implement it fully.

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, the best practice is already implemented reasonably well without input from the library's user.

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 datasets 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 [23].

[2] Reported state-of-the-art and state-of-practice practices from Tables 2, 3, and 4 of [24].

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 and its documentation together[3]	Dashboard and Swagger	✓✓
Parallelise feature extraction[3]	`parallel_map`	✓✓
Cache predictions[3]	`@GreatAI.create`	✓✓
Allow robustly composing inference functions[3]	All decorators support async	✓✓
Implement standard 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 [24].

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

In Table 6.2, we added six additional best practices, which are generally well-known software engineering considerations that are also applicable to AI/ML deployments. These had 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.

Quantifying the number of implemented best practices would be misleading since their scope and importance cover a wide range; furthermore, there is some overlap between the different studies and even within the studies. However, it is still clear that a large number of best practices (17) can be given a *Fully automated* implementation by *GreatAI*'s design, and many others (16) 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 while still implementing various best practices (**RQ2**).

6.2 Interviews

One of the central takeaways of Section 2.3 is that, for example, Seldon Core is useful for implementing or helping to implement most of the best practices. Regardless, it also has an initial threshold that must be surmounted before implementing even a single one. According to the adoption rate surveys, this may discourage a large portion of practitioners from using it or other similar frameworks. The presented solution offers a different mix of features: the initial threshold is virtually non-existent; hence, best practices can be applied immediately. But at the same time, it only covers a more limited range of practices.

Our hypothesis is that the latter approach aligns better with the expectations of professionals. To verify this, we conducted a series of interviews with the cooperation of ten industry practitioners with varying levels of Software Engineering (SE) and Data Science (DS) experience. In this section, the question of generalisability (**RQ4**) is investigated using the interview methodology described in Section 3.2. The participants were gathered through the recommendations of my friends and colleagues. All of the final interviewees have had at least some expertise in both DS (with a median of 2.5 years) and SE (with a median of 2 years).

6.2.1 Best practices survey

The practitioners were first asked to fill out a questionnaire about their latest AI/ML project involving deployment. This point-in-time measurement (shown in Appendix A) served as a baseline for the deployment quality they are used to. Analysing the results show that the amount of software engineering experience has a moderately strong correlation ($r_{Pearson} = 0.67$ with $p = 0.0033$) with the overall number and extent of implemented deployment best practices. This is illustrated in Figure 6.1. Interestingly but unsurprisingly, there is no similar statistically significant relationship regarding the amount of data science experience.

The y-axis of Figure 6.1 is calculated by discarding the *Not applicable* answers and projecting the 5-point Likert scale to a range from 0 to 1, which is subsequently averaged over all questions. The overall mean adoption rate/extent is just above 0.5, which equates to the *Neither agree nor disagree* label. These data are in line with the findings of Serban et al. [4].

Because the survey's 15 questions were compiled from the *Fully automated* rows of Tables 6.1 and 6.2, that means that when using *GreatAI*, they are all implemented automatically. Consequently, the adoption rate/extent is doubled immediately just by wrapping the inference function with `@GreatAI.create`: this

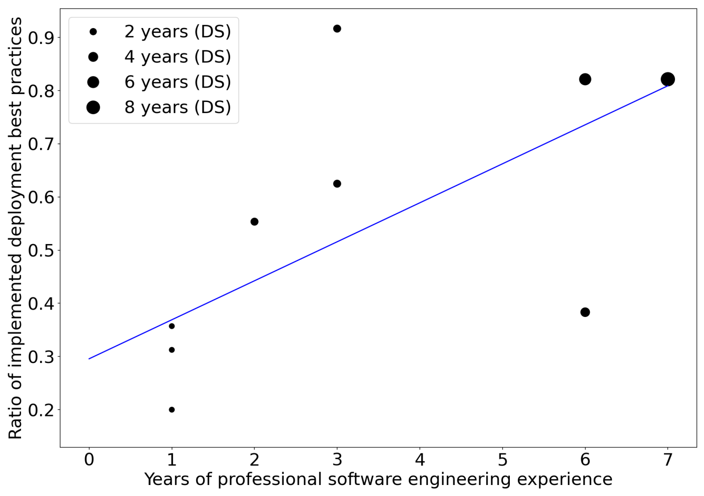


Figure 6.1: Best practices adoption rate as a function of Software Engineering (SE) and Data Science (DS) experience. SE experience is shown on the horizontal axis, while the point sizes denote the practitioners' experience in DS. The correlation between the axes is significant ($r_{Pearson} = 0.67$ with $p = 0.0033$).

is the added value of *GreatAI*[1]. Moreover, this provides further evidence for answering **RQ3** showing the extent of automatically implemented practices over non-*GreatAI* deployments.

6.2.2 Technology acceptance

Table 6.3: Technology acceptance model survey (presented in Appendix B, sample size = 10) results per variable. The input values range from 1 to 7.

	Perceived ease of use	**Perceived utility**	**Intention to use**
Median	5.8	6.4	6.3
Mean	5.5	6.1	6.0
Standard deviation	1.0	0.9	1.3
Cronbach's alpha	0.77	0.88	0.95

The participants filled out a form (shown in Appendix B) after finishing their first deployment with *GreatAI* to provide data for creating the technology acceptance model of the problem context. The survey contained ten questions from three categories, which could be rated on a 7-point Likert scale. The summary of the answers is presented in Table 6.3. The high Cronbach's alpha values indicate strong internal consistency [94] for each TAM dimension; thus, averaging the responses per category is semantically meaningful.

Following the methodology of [38], the connections between the Perceived Utility (PU), Perceived Ease Of Use (PEOU), and Intention To Use (ITU) dimensions of TAM were analysed. Two statistically significant ($P \leq 0.05$) correlations were uncovered: between PU and ITU ($r_{Pearson} = 0.81$ with $p = 0.0048$); and PEOU and ITU ($r_{Pearson} = 0.80$ with $p = 0.0068$). Learning from the findings of prior case

[1]As explained earlier, measuring quality as a function of best practice count would be dubious. Thus, the achieved magnitude of the doubling is irrelevant; however, the direction of change is not.

studies, it is reasonable to believe that both the *perceived utility* and the *perceived ease of use* play an equally important role in influencing professionals' *intention to use* the deployment framework.

The assessment of *ease of use* lags behind the rest, but it is still quite high. It may be possible that PEOU would go up with further use. Nevertheless, the high *perceived utility* implies that *GreatAI* shows its value early on. This, combined with the correlations uncovered within the context's technology acceptance model, validates the hypothesis that focusing on good API design is just as necessary as providing practical features.

6.2.3 Task solving & exit interviews

In order to give qualitative depth to the previously presented quantitative results, it is time to discuss the main segment of the interviews. The participants' backgrounds covered a vast and fascinating cross-section of industrial AI/ML. The financial sector was represented by a researcher working on market prediction models for the Hungarian State Treasury and two people building an upcoming digital bank's core services. Image processing contexts were illustrated by professionals predicting Sun activity at the European Space Agency and different ones creating pose-recognition at a startup for people with disabilities using 3D cameras. Moreover, investigating companies' AI use as part of due diligence processes and intrusion detection from network packet traces are just some of the other core activities the interviewees had been doing recently.

Stemming from this diversity, these semi-structured interviews could be expected to provide valuable insights into the generalisability of *GreatAI*. The methodology of Section 3.2 was followed by applying reflective journaling and thematic analysis. After labelling each aspect of the feedback, and two iterations of merging redundant or related topics, we ended up with three overarching themes: *Functionality*, *API*, and *Responsibility to adopt*. As we will soon see, these correspond to the *perceived utility*, *perceived ease of use*, and *intention to use* components of TAM fairly well.

Functionality The library's feature-set was complimented during most interviews, with one participant noting that, although the overall number of features is relatively small, most of them are utilised in most cases. Similarly, the `utilities` submodule was appreciated for helping greatly in the interview task, but non-NLP researchers noted its likely inadequacy for their area. Still, they would like to see "bundle" or "toolbox"-style modules for their fields because it would save them from a lot of copy-pasting.

The effortless parallel feature extraction and large file handling support were highlighted multiple times for the reason that the particular interviewees had not encountered other libraries providing these features. Other concrete features, such as the searchable *exceptions* column in the Dashboard's table and the *feedback* mechanism, were also popular. One professional highlighted the latter for coercing users to consider a human-in-the-loop approach which was said to be often expected in modern systems.

When reflecting on the framework from a bird's eye view, the generality and extensibility of the API were emphasised. As explained by a senior engineer, this is mainly because once you commit to using it, it is important not to find yourself at a dead end for a specific use case forcing you to look for a different library. However, two participants also noted that for complete generality, `MATLAB` support would be necessary. Regarding non-functional features, private hosting (especially in banking and government), open-source auditability, and good scalability (by means of an external database) were the top subjects of praise.

API Regarding the surface through which clients interact with the library, the feedback is also positive but more nuanced. Many participants liked that the functions' behaviour is easy to guess from their names. The decorator syntax caused minor confusion but consulting the documentation solved the issues in all three cases. The CLI app `great-ai` was appreciated for having a close to trivial signature; the participant noted that she strives to use as few CLI commands as feasible. Surprisingly, even the practitioners with more data science background appreciated the Docker support. Nonetheless, one expert had a feature request for a configuration GUI because his colleagues are used to handling MATLAB App Designer applications.

The recurring theme of the discussions focused on the question of "*How simple is too simple?*". The argument is that an API cannot be simpler than the domain in which it exists. More precisely, it can only be simpler at the cost of losing transparency. Let us take the example of saving models using `save_model()`. If a project is set up correctly, it either has an initial `configure()` call to the storage provider backend, or it has an appropriately named credentials file in the project's root, for instance, `s3.ini` or `mongo.ini`. Once set up, it is trivial to use as long as we do not divert from the happy path. However, if an issue arises, such as an upgrade or migration of MongoDB, debugging the application is non-trivial for its lack of transparency.

In other words, we could say that the average (cognitive) complexity is low while the worst-case is as high — if not higher — than without using `save_model()`. This proved to be somewhat controversial. However, ultimately, optimising the happy path of the AI/ML development lifecycle was deemed worthwhile by the participants in most cases. With the argument that the majority of the time spent during a project is spent on this path anyway. However, this raises the question of who exactly are the target users of *GreatAI* and who will fix arising issues?

Responsibility to adopt Let us first look at some insightful anecdotes that surfaced during the interviews. Especially in more research-oriented environments, production deployment pipelines can be of questionable robustness. This phenomenon was demonstrated by one account of a simple single-machine deployment pipeline: it is an interplay of `cron` jobs calling a series of shell and MATLAB scripts resembling a Rube Goldberg machine. But connecting a couple of Google Colab accounts to a GitHub repository and Weights&Biases[2] to implement parallel model training can also be found in the wild.

Moreover, various research companies were mentioned that for multiple years used to or still have an R&D department consisting solely of data scientists. In one extreme case, the staff was described as more than 30 data scientists and 0 other technical employees. In such a setup, it is unreasonable to expect even professionals to have the capabilities and focus to set up the required foundation for handling all best practices. All but one interviewee verified this assumption. They also referred to their previous projects, which usually required many researchers and experts from various fields, and too often, software engineers had not been prioritised to be included.

Doing software engineering without software engineers is difficult. *GreatAI* is not a viable replacement for any well-trained expert, though it is still better than nothing. During the interviews, we realised that the likely underlying reason for not employing AI engineers or software engineers as part of AI/ML projects is a lack of awareness. This was theorised by some and demonstrated by six participants who had, even though followed some, not explicitly sought out information on AI deployment best practices. Thus, raising awareness — especially by presenting a value proposition, e.g. lower maintenance costs and better long-term quality — might be crucial for improving AI deployments in general. Verifying this hypothesis could be a worthwhile direction for future research.

During the larger discussions, *GreatAI* was deemed appropriate for raising awareness since it showcases how even a simple library is able to implement a lot of best practices. Additionally, it was noted that it could also be considered for one-person projects where — by definition — it is admissible to have no SE

[2]wandb.ai

expert on the "team". To further help such cases, integrating a one-click Heroku[3] app deployment was also recommended to simplify the entire last portion of the lifecycle.

6.2.4 Discussion of interviews

The overall takeaway from this is that most features were well-received, and the high mean value of *perceived utility* is credible. The criticism of being NLP-centric is also justified: the initial scope of the proof-of-principle framework was limited to this domain. Nonetheless, learning the experts' opinion that they wish to have a similarly specific solution to their problem contexts is reassuring because it proves that the API is not only generalisable but is expected to be generalised. At the same time, it is crucial to admit that no one-size-fits-all solution can exist for such a diverse domain. Therefore, allowing customisability and easy extension of the system must remain central design questions.

Regarding the API's level of abstraction, we have to agree with the experts that the problem of deployment cannot be "magically" solved by a trivial API. However, solving deployment problems can be streamlined, at least in simpler cases. At the same time, the complex ones can be left to the professionals with relevant knowledge. This parallels the AI-libraries that have inspired *GreatAI*. For instance, Hugging Face `transformers` streamlines fine-tuning and applying SOTA models, but it does not provide any facilities to help you create the next SOTA architecture because that is a vastly more complex task that most users are not expected to tackle.

In order to reach its goal of improving best practice adoption, *GreatAI* can help raise awareness by presenting a verifiable value proposition, i.e. a couple of lines of code can already result in more maintainable, robust, high-quality deployments. This might prompt users or technical decision-makers to invest more in software engineering in AI/ML projects. Additionally, it can help the effectiveness of AI/software engineers by handling the grunt work of implementing some best practices, leaving them with more resources to focus on the complex and creative aspects of *GREAT* deployments.

In summary, the answer to *How suitable is the design of GreatAI for helping to apply best practices in other contexts?* (**RQ4**) is — unsurprisingly — subjective. Combining the high value of *intention to use* from Table 6.3, the generally positive feedback regarding the library's added value, and the numerous feature requests for fitting it to specific needs, we conclude that there is some chance of suitability for generalisability. The existence of this potential is already exciting and presents an opportunity for experimenting with building on the design of *GreatAI*.

6.2.5 Threats to validity

Two potential threats to the validity of the experiments and their results are identified. Firstly, the claimed utility of the framework derived in Subsection 6.2.1 does not take into account the practical significance of the implemented features and, therefore, may be subject to bias. However, the *perceived utility* evaluations indicate that the participating engineers and scientists identify practical value in the features of *GreatAI*. Nevertheless, in the future, we intend to extend the range of implemented best practices, which would in turn, give higher confidence about the achievable quantitative improvement through using the library.

Secondly, the survey answers and, in general, the interviewees may be subject to bias. The small sample size of practitioners can reasonably lead to some groups being over- or under-represented. The presence of selection bias is also plausible. These could be mitigated by gathering more data in future research. Coming from the exploratory nature of this analysis, many insights could be gained from the collected data. However, for confidently generalising the results, more data are needed.

[3]heroku.com

6.3 Future work

The primary purpose of the library 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[4].

The library's main limitations come from its bias toward NLP deployments. This is not unreasonable given the design's exploratory nature and the context of the case studies. Nevertheless, future work must focus on introducing and balancing support for many more fields' deployments. Although *GreatAI* has already proved its utility, it has also shown that generalising and extending its functionality would be worthwhile. Therefore, many potential improvements are presented below.

6.3.1 More ML domains and modalities

The cases presented in Chapter 5 revolved around NLP. This, of course, 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 (which caters to textual and tabular data), while the latter gives robust feature extraction support only for textual input. However, in practice, sound, image, and video are also widely taken as input. Furthermore, with the rise of multimodal models [95], even different combinations of them may be simultaneously taken as input.

Supporting the easy, direct upload of larger non-JSON files — e.g. by saving them to S3 and showing a preview of them on the Dashboard's traces table — and extending `utilities` to handle multimedia formats should be sufficient for counteracting the NLP bias. Hence, widely expanding the scope of applicability of *GreatAI*. As we have seen in Section 4.4, the architecture is otherwise adequately general; therefore, incremental extensions can be applied.

6.3.2 More best practices

In order to greatly simplify its API, each *GreatAI* Trace is a single document with a well-defined 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 environment, storing Traces in that can be favourable. PostgreSQL [96] is a popular choice, and it also features good JSON document support. Hence, introducing first-class integration for PostgreSQL could benefit some clients.

Data-intensive services can fall into three broad categories: online systems, batch processing, and stream processing (near-teal-time systems) [54]. 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 [97], AWS SQS [98], or AMQP [99]. Moreover, some metrics of *GreatAI*, such as the cache statistics, versions, and derived data from traces, can already be conveniently queried from its REST API. Nevertheless, adding support for the de facto standard metric gathering tool, Prometheus[5], could save the library's users from one more integration step.

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

[4]Available at pypi.org/project/great-ai and hub.docker.com/repository/docker/schmelczera/great-ai.

[5]prometheus.io

Chapter 7

Conclusion

Concerned by the asymmetry between the industry's adoption of accessible AI/ML-libraries and existing solutions for their robust deployment, we investigated this phenomenon's causes and potential resolution. When looking at various recent case studies, a recurring theme was revealed: *transitioning* from prototypes to production-ready AI/ML deployments is a source of adversity for small and large enterprises alike. Even though several frameworks and platforms exist for facilitating this step, surveys on the execution of best practices continue to expose the industry's shortcomings. This signals that existing libraries are underutilised, which may lead to poor deployments that underperform or develop issues that go unnoticed and might inflict societal harm.

We hypothesised that presenting a library which implements best practices and is also optimised for ease of adoption could help increase the overall quality of industrial AI/ML deployments. To test this, we designed and implemented a framework based on the principles of cognitive science and the prior art of software design. Subsequently, we tested and refined the design in an iterative process. First, we developed and deployed a model for classifying the domains of academic publications. Then, we fine-tuned and deployed a SciBERT model for generating publications' technology-transfer summaries. *GreatAI* had been proven helpful; therefore, after feeding back the insights gained into its design, we turned it into an open-source library. Furthermore, *GreatAI* has been successfully integrated into every production deployment of ScoutinScience since then and receives thousands of monthly downloads.

During the refinement of the framework, six previously unaddressed AI/ML deployment best practices were identified. Including these, the framework fully implements 17 best practices while it provides support for another 16. We validated the value provided by implementing or helping to implement these practices through interviews with ten industry professionals from various subfields.

The interview participants completed two questionnaires, the results of one of which indicated that using *GreatAI* in an example task increased the number of implemented best practices, on average, by 49% compared with their latest project. We also calculated the technology acceptance model of the context; a significantly strong correlation was measured between the *perceived ease of use*, the *perceived utility* and the *intention to use* dimensions. Overall, proving that ease of use is just as important as core functionality when adopting AI deployment frameworks.

The open-ended exit interviews revealed that value can be derived from the library even in its current form and that the API's design has the opportunity to generalise to other fields of industrial AI/ML applications. However, they also highlighted that adoption issues do not necessarily come from a lack of willingness but a lack of awareness. Even if the returns achievable from good deployments are well worth the investment. Nevertheless, this value proposition needs to be conveyed and proved to data science professionals and technical decision-makers; and *GreatAI* might just be the ideal candidate for doing that.

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 needed for robust and improvable deployments. At the same time, it also saves them from the menial work of manually implementing these constructs. While most importantly, it proves that increasing the adoption rate of AI/ML deployment best practices is feasible by designing narrower and deeper APIs.

Good deployments benefit all of us. Accordingly, continued research into the means of good deployments remains crucial. However, next to that — as the presented results have shown — better deployments can also be achieved by facilitating the *transition* step of the AI lifecycle with a focus on adoptability. Having automated implementations, even if for just the straightforward best practices, leaves professionals additional time to tackle the more complex deployment challenges and fewer opportunities to miss critical steps. Overall, resulting in more general, robust, end-to-end, automated, and trustworthy AI deployments.

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] 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)

[13] Russell S. J.: Artificial intelligence a modern approach. Pearson Education, Inc. (2010)

[14] 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)

[15] 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)

[16] 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)

[17] 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)

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

[19] 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)

[20] 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 (2016)

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

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

[23] 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)

[24] 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)

[25] 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)

[26] 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)

[27] 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)

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

[29] 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)

[30] 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)

[31] 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)

[32] 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)

[33] 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)

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

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

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

[37] Alhojailan M. I.: Thematic analysis: A critical review of its process and evaluation. West east journal of social sciences: vol. 1(1), pp. 39–47 (2012)

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

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

[40] 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)

[41] 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)

[42] Wu C. S., Cheng F. F., Yen D. C., Huang Y. W.: User acceptance of wireless technology in organizations: A comparison of alternative models. Computer Standards & Interfaces: vol. 33(1), pp. 50–58 (2011)

[43] Bland J. M., Altman D. G.: Statistics notes: Cronbach's alpha. Bmj: vol. 314(7080), p. 572 (1997)

[44] Hynes N., Sculley D., Terry M.: The data linter: Lightweight, automated sanity checking for ml data sets. In NIPS MLSys Workshop: vol. 1 (2017)

[45] Bishop M.: Robust programming. handout for (1998)

[46] Ritchie D. M., Thompson K.: The UNIX timesharing system. Bell System Technical Journal: vol. 57(6), pp. 1905–1929 (1978)

[47] Salus P. H.: A quarter century of UNIX. ACM Press/Addison-Wesley Publishing Co. (1994)

[48] Ousterhout J. K.: A philosophy of software design: vol. 98. Yaknyam Press Palo Alto (2018)

[49] Hermans F.: The Programmer's Brain: What every programmer needs to know about cognition. Simon and Schuster (2021)

[50] 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)

[51] 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)

[52] Deissenboeck F., Pizka M.: Concise and consistent naming. Software Quality Journal: vol. 14(3), pp. 261–282 (2006)

[53] 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)

[54] Kleppmann M.: Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc." (2017)

[55] Procida D.: Diátaxis documentation framework. URL https://diataxis.fr/

[56] 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)

[57] Martin R. C.: Clean code: a handbook of agile software craftsmanship. Pearson Education (2009)

[58] Rumbaugh J., Jacobson I., Booch G.: Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education: ISBN 0321245628 (2004)

[59] Wang L., Wang C.: Producer-consumer model based thread pool design. In Journal of Physics: Conference Series: vol. 1616: p. 012073: IOP Publishing (2020)

[60] Maron M. E.: Automatic indexing: an experimental inquiry. Journal of the ACM (JACM): vol. 8(3), pp. 404–417 (1961)

[61] Beltagy I., Lo K., Cohan A.: SciBERT: A pretrained language model for scientific text. arXiv preprint arXiv:1903.10676 (2019)

[62] 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)

[63] 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)

[64] Lo K., Wang L. L., Neumann M., Kinney R. M., Weld D. S.: S2ORC: The Semantic Scholar Open Research Corpus. In ACL (2020)

[65] 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)

[66] Buckley C.: Implementation of the SMART information retrieval system. Tech. rep.: Cornell University (1985)

[67] Hand D. J., Yu K.: Idiot's Bayes—not so stupid after all? International statistical review: vol. 69(3), pp. 385–398 (2001)

[68] 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)

[69] 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)

[70] 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)

[71] Mikolov T., Chen K., Corrado G., Dean J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)

[72] 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)

[73] Maynez J., Narayan S., Bohnet B., McDonald R.: On faithfulness and factuality in abstractive summarization. arXiv preprint arXiv:2005.00661 (2020)

[74] Berkovsky S., Baldwin T., Zukerman I.: Aspect-based personalized text summarization. In International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems: pp. 267–270: Springer (2008)

[75] 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)

[76] 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)

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

[78] 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)

[79] 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)

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

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

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

[83] 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)

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

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

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

[87] 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)

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

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

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

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

[92] 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)

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

[94] Nunnally J. C.: Psychometric theory 3E. Tata McGraw-hill education (1994)

[95] Gao J., Li P., Chen Z., Zhang J.: A survey on deep learning for multimodal data fusion. Neural Computation: vol. 32(5), pp. 829–864 (2020)

[96] Momjian B.: PostgreSQL: introduction and concepts: vol. 192. Addison-Wesley New York (2001)

[97] 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)

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

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

Appendix A

Best practices assessment

Similarly to the approach of [4], participants are asked about their team's level of adoption of AI/ML deployment best practices. 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 collaborated on implement 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 datasets 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 a 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 and its documentation together
13. Parallelise feature extraction
14. Cache predictions
15. Allow robustly composing inference functions

Appendix B

TAM questionnaire

Following the methodology for the parsimonious technology acceptance model of Wu et al. [42], 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.