Teaching Reproducibility and Embracing Variability: From Floating-Point Experiments to Replicating Research

Mathieu Acher, Arnaud Gotlieb, Helge Spieker, Gauthier Le Bartz Lyan

Teaching Reproducibility and Embracing Variability: From Floating-Point Experiments to Replicating Research

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France
mathieu.acher@inria.fr

Arnaud Gotlieb
Simula Research Laboratory
Oslo, Norway
arnaud@simula.no

Helge Spieker
Simula Research Laboratory
Oslo, Norway
helge@simula.no

Gauthier Le Bartz Lyan
Inria
Rennes, France
gauthier.le-bartz-lyan@inria.fr

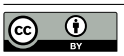
Abstract

Reproducibility is often discussed but rarely practiced in undergraduate computer science education. In this paper, we present the design, implementation, and evaluation of a 24-hour hands-on course entirely dedicated to reproducibility and variability in computational experiments. Taught to fourth- and fifth-year students at INSA Rennes in Fall 2024, the course combines scientific thinking, software engineering practices, and variability analysis. Students first explored the non-associativity of floating-point arithmetic as a reproducibility "Hello World" using Docker, GitHub Actions, and templated experimentation to analyze sources of variability across programming languages, compiler flags, and numerical precision. The second half of the course focused on reproducing and replicating actual research papers, including studies on large language models playing chess, home advantage in football during COVID-19, and energy efficiency across programming languages. Students successfully reproduced key results, identified subtle reproducibility issues such as changes in library defaults, and designed replications that extended or challenged original findings. We describe the course structure, pedagogical strategies, and lessons learned, including when students found reproducibility flaws in the instructor's own prior work. Our experience suggests that reproducibility and variability deserve a central place in computer science education and can be taught in a way that is both technically rigorous and scientifically engaging.

CCS Concepts

• **Software and its engineering → Software product lines**; • **Computational science and engineering education**;

ACM Reference Format:
Mathieu Acher, Arnaud Gotlieb, Helge Spieker, and Gauthier Le Bartz Lyan. 2025. Teaching Reproducibility and Embracing Variability: From Floating-Point Experiments to Replicating Research. In *ACM Conference on Reproducibility and Replicability (ACM REP '25), July 29–31, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3736731.3746162



ACM REP '25, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1958-5/2025/07
https://doi.org/10.1145/3736731.3746162

1 Introduction

Reproducibility is a fundamental aspect of scientific and engineering integrity. In scientific disciplines, it ensures that findings are independently verifiable and generalize; in engineering, it guarantees that systems behave predictably across deployments. Reproducibility and replicability are also ways to innovate and explore different hypotheses or ideas. Another related observation is that software increasingly underpins scientific discovery and industrial innovation – from climate modeling to autonomous systems – and is key to mastering reproducibility. Modern research and industry are actually relying on intricate software systems to derive insights from data, simulate phenomena, and automate decision-making. For example, climate science combines mathematical models, large-scale simulations, and data mining pipelines, all built on top of layers of software infrastructure [8, 14]. These systems are not merely tools, but artifacts of engineering—subject to design choices, dependencies, and environmental conditions that introduce variability. Whether in scientific experiments or production deployments, seemingly minor differences in software versions, configurations, or runtime environments can lead to divergent results [7, 16, 23].

This challenge extends beyond pure scientific discovery. In industry, non-reproducible builds or unreliably trained machine learning models incur costly technical debt [11]. Even when code and data are available, studies in neuroimaging, machine learning, and software engineering reveal that "re-running" an analysis often fails to reproduce the original results [4, 10, 18]. Such inconsistencies undermine the the trust in computational findings and obscure the root causes of discrepancies, whether they arise from algorithmic assumptions, software layers, or inherent uncertainty in the problem domain.

Reproducibility, then, is not just a scientific ideal, but a lens for critical inquiry. It demands that engineers scrutinize their tools, document their environments, and interrogate why systems behave differently under varying conditions. For computer science students, mastering these skills is essential – not only to validate results but to navigate the complexities of real-world software systems and domains, where variability is the rule rather than the exception.

Although reproducibility is at the heart of computer science, covering as diverse areas as systems design and statistical analysis, it is rarely the primary focus of courses. Its adoption in computer science education is still in its early stages, and reproducibility is rarely taught as a unified discipline. Instead, its principles can be

often scattered across courses: a mention in a research methods lecture, a tool demo in a software engineering lab, or an afterthought in a data science project. In 2024, we took the opportunity to fill in the gaps in the curriculum and built a course entirely dedicated to reproducibility. This paper presents a hands-on, self-contained course in reproducibility and variability, taught in Fall 2024 to fourth- and fifth-year students in computer science at INSA Rennes. The course was motivated by three key objectives. First, reproducibility is a fundamental property of software systems that connects directly to modern *software engineering* practices. Students must master tools and techniques from continuous integration, DevOps, automated testing, and benchmarking to ensure reproducible results. Second, even for students who do not pursue research careers, understanding and applying *scientific method* is crucial. This includes reading and analyzing research papers, formulating research questions, designing experiments, and effectively communicating findings. Third, students must understand and practice managing (software) variability, from identifying variation points and threats to validity to conducting systematic ablation studies at the software level. Hence, *variability management* is essential for both reproducibility (finding the right computational environment) and replicability (testing results under controlled variations).

The course structure emphasizes these objectives through practical exercises and real-world case studies. Students gain experience with reproducibility tools while developing a deeper understanding of how variability affects computational results. This approach bridges the gap between software engineering practices and scientific methodology, preparing students for careers where computational reproducibility is increasingly important.

The contributions of this paper are as follows:

- We present a reproducibility course structure that emphasizes the role of software engineering, variability and experimental design, based on two complementary parts;
- We demonstrate how a simple numerical question—associativity of floating-point addition—can become a rich case study for exploring software tooling and variability;
- We describe a methodology for teaching reproduction and replication of research papers;
- We reflect on lessons learned, including unexpected outcomes of student-led discoveries of variability sources and the robustness (or brittleness) of published results.

2 Design of a Reproducibility Course

2.1 Reproducibility Deserves Its Own Course

Despite its importance in scientific research, software engineering, or data analysis, reproducibility is rarely taught as a central, standalone topic in computer science curricula. Instead, reproducibility tends to appear in fragmented forms, either briefly mentioned in research methods classes, practiced informally in software engineering labs, or touched upon during project-based work. This fragmented exposure is not ideal for developing the deep and critical understanding needed to handle reproducibility in real world settings.

Another trend is that the increasing complexity of computational science calls for mastering software and its engineering. From climate simulations and bioinformatics pipelines to machine learning experiments and data-driven social science, the reliability of results depends on the ability to reproduce and understand numerous variability factors within software artifacts. Students entering these fields need structured training to deal with configuration management, tool versioning, stochastic behavior of software systems, etc.

At INSA Rennes, we identified an opportunity to fill this curriculum gap by designing a reproducibility course from the ground up. The aim was to develop student competencies not just in tools, but in thinking reproducibly: questioning results, managing experimental variability, and practicing science. We observed that students might not have experience in reading and understanding scientific papers or thinking about research questions and how experiments are designed. By dedicating a full course to these goals, we could build a coherent progression from intuition to software automation, from simple examples to real-world replications, and from tool usage to scientific discovery or reflection.

2.2 Pedagogical Objectives

The course was designed around three main pedagogical pillars.
Scientific method practice: Students learn to think like researchers (reading and analyzing articles, formulating hypotheses, designing experiments and interpreting results, etc.). Even students not pursuing research careers benefit from this analytical rigor.
Software engineering: Reproducibility is directly related to modern software practices, including version control, continuous integration, containerization, and infrastructure automation. Students apply these tools in lab sessions to develop reproducible pipelines.
Variability awareness/management: Students explore the multiple dimensions of variability in software systems and computational experiments (input data, implementation details, environment, randomness, etc.) and learn to reason about them systematically.

2.3 Course Structure and Organization

The course was offered in Fall 2024 to students in the fourth and fifth year of their computer science engineering program at INSA Rennes. The cohort of 20 students had solid foundations in programming and software tools but limited exposure to the scientific method or reproducibility as a formal discipline. The course consisted of six 4-hour sessions and a 2-hour session (26 hours total), alternating between mini-lectures, guided labs, and open-ended experimentation. The curriculum included:

- **Introduction to Reproducibility (2h):** Definitions and distinctions between reproducibility, repeatability, and replicability; historical context; importance in science and industry; overview of reproducibility initiatives and crises.
- **Exploratory Experiment (2h):** A hands-on dive into floating-point associativity, prompting students to explore the non-associativity of $x + y + z$ in practice and uncover variability across inputs, languages, and environments.
- **Tooling for Reproducibility (4h):** Introduction to Git, Docker, and CI (GitHub Actions); Students containerize experiments and reproduce each other's work in shared repositories.

- **Modeling and Analyzing Variability (4h):** Students identify, implement, and analyze variability factors using templates, CLI flags, and parameter sweeps. Jupyter and pandas are used to structure and analyze the experimental data.
- **Reproducing a Research Paper (4h):** Teams reproduce the results of a published study using the original code and data, ensuring automation and documentation.
- **Replicating and Extending Research (10h):** Controlled experiments to replicate the study under variation. Students explore generalizability and reflect on the robustness of scientific claims.

2.4 Teaching Approach and Philosophy

The course emphasized hands-on inquiry, peer collaboration, and scientific reasoning. Rather than present reproducibility as a checklist of tools or guidelines, we approach it as a mindset and methodology. Key features of the pedagogy included the following.

Learning by doing: Students explore variability through experiments, not lectures. A "Hello World" problem becomes the gateway to modeling complex systems.

Collaboration and peer reproduction: Students used to work in pairs (forming "groups") to encourage discussions, collaborations and sharing of the work. The students also reproduced each other's results (across groups), revealing the hidden assumptions or the absence of an established truth in experimental design.

Tooling as infrastructure, not Goal: While students learned Git, Docker, and CI/CD pipelines, these were framed as enablers of reproducible thinking, not ends in themselves.

Critical engagement with scientific work: By reproducing and replicating real research papers, students learned to navigate incomplete documentation, shifting libraries, and implicit assumptions, thus gaining insight into the realities of research reproducibility.

2.5 Related Work

Fund argued that reproducibility should be emphasized throughout the computer science curriculum to promote better research practices from the outset [9]. Our context is unique in that it allows for the development of a dedicated course on reproducibility. This opportunity may not be feasible in many other curricula. This enables us to give the topic focused attention while also connecting it to key areas such as software engineering and data science. Techniques introduced in the reproducibility course (such as continuous integration) can be further explored in these specialized domains. Importantly, reproducibility is also addressed in other advanced courses at INSA Rennes, such as compilers and the design of domain-specific languages, where students run benchmarks and engage with issues of experimental rigor and repeatability. From this perspective, material from the reproducibility course can stand alone and be adapted to enrich other parts of the curriculum.

There are initiatives to challenge practitioners and students in reproducing papers. For example, ReproducedPapers.org structures machine learning reproducibility as an open and accessible educational resource [24]. Reprohackathons provide a collaborative training format to promote reproducibility in bioinformatics [6]. Reproducibility-focused training in neuroimaging supports the development of open and robust computational workflows [19]. Large-scale reproducibility verification can be integrated into statistics education to teach robust scientific inquiry [22]. The risk is that some instructors may not be familiar with the considered papers. A major lesson learned (see below) is that deeply knowing papers to reproduce (e.g., being involved in the original paper) is important to foster collaborations and exchanges.

Reproducibility tasks can effectively introduce core AI ethics principles such as fairness and accountability [15]. Although mentioned, we do not integrate such important topics in this course.

Kiar el at. [13] call to embrace experimental variation in neuroimaging. We follow a similar attitude through variability management and replication. Reproducibility engineering can be considered a critical lifelong skill in software engineering education [17]. We concur and connect as much as possible software engineering tools and methods to reproducibility.

3 A Hello World for Reproducibility: $(x + y) + z$

We introduce the associativity of floating-point addition as a deceptively simple problem that invites deep exploration. The question "How often is $(x + y) + z = x + (y + z)$?" prompted discussion of numerical precision, implementation details, variability, and reproductibility. It is also a way of practicing basic software tools that have proved to be useful in the context of reproducibility.

3.1 Challenging Students with a Simple-Looking Problem

To kick off the practical component of the course, we deliberately chose a task that looks trivial on the surface but hides a rich set of challenges. The students were given the task of evaluating whether floating-point addition is associative with random values: that is, checking whether $x+(y+z) = (x+y)+z$ holds or not. Although many expected this identity to be satisfied by default, students quickly discovered that numerous counterexamples exist and that it fails for a significant fraction of inputs. The original yes/no question then becomes a quantitative question, and we can find how often the associative property holds.

This task served as an excuse to investigate deeper the possible sources of the discrepancy. Students explored how this proportion changed significantly based on seed values (input data variability), how results varied between programming languages and libraries (language and library variability), and how compiling the same program on different platforms or processors affected results (platform and processor variability).

Rather than immediately explaining these behaviors, we challenged students to discover, model, and explain them through structured experimentation. This inquiry-based process helped students connect seemingly minor implementation choices to broader reproducibility issues.

We used this example to transition from intuition-driven exploration to rigorous, automated experimentation. The students implemented reproducible workflows using Git, Docker, and GitHub Actions. They then shared and reproduced each other's findings, exposing them to the challenges of aligning results in the presence of environmental and implementation variability. It also calls for partly automating the analysis of data – including the gathering

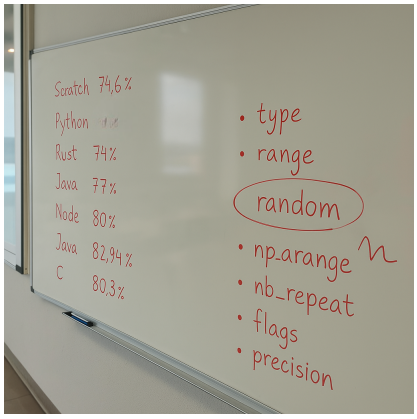


Figure 1: Live results and discussions about variability factors

and structuring of data as well as statistical analysis (e.g., based on machine learning) to extract some patterns – and the interpretability of results. The associativity task became a central pedagogical anchor for the course. It allowed students to practice reproducibility tooling and confront the nuanced ways in which variability manifests itself in computational settings.

3.2 From Intuition to Automation

Specifically, we started with the provocative question: Is $x+(y+z) = (x+y)+z$? The semantics of equality, the "type" of x, y, and z, and the semantics of "+" can be part of the discussion, but students quickly engage with the question. It has limited interest in the case of integer values, and quickly the focus is on floating-point values. Some students can be surprised by the non-associativity of floating-point computation, and try to laboriously find a counterexample. Some students have heard about the issue and try to rediscover an example (e.g., with 0.1+0.2+0.3). In any case, we let students conduct exploratory experiments in their chosen languages (e.g., Python, Java). We then move on to quantify how often the associative property holds.

After some time, the instructor collects lively results on a whiteboard. Figure 1 shows an excerpt with the languages used associated with a percentage on the left-hand side. Clearly, the results are not consistent. The students are not to blame here, since we are scratching the surface. However, it is a starting point to investigate the possible root causes in a more detailed and rigorous way. The first hypothesis is that it comes from the programming language itself. Then other hypotheses can be formulated: effect of randomness, effect of types (e.g., float vs long in Java), effect of repetitions, etc.

3.3 Variability Exploration

The students modeled, implemented, and analyzed variability factors such as language, compiler flags, system architecture, and floating-point precision. Each can subtly or significantly influence the outcome of a computation (here a percentage), particularly when evaluating properties such as associativity in floating-point arithmetic.

To explore the impact of variability factors on numerical reproducibility, students could take advantage of three main approaches presented in the lab session: a template-based code generation

```
import random
import math
import numpy as np


def check_property(repetitions):
    correct_count = 0
    for _ in range(repetitions):
        x = {{ op_creation }}
        y = {{ op_creation }}
        z = {{ op_creation }}

        # Check if the associative property holds
        result1 = ((x + y) + z)
        result2 = (x + (y + z))

        if {{ op_compare }}:
            correct_count += 1

    print(f"{correct_count / repetitions * 100:.2f}%")


# Define the number of repetitions
check_property(int(math.pow(10, 6)))
```

Figure 2: Template-based approach and parameterization

```
from jinja2 import Template

template_content = open('property_template.py.jinja').read()
template = Template(template_content)

factors = {
    'op_creation': {
        "random": "random.random()",
        "uniform": "random.uniform(0, 1)",
        "uniform10": "random.uniform(0, 10)",
        "uniform100": "random.uniform(0, 100)",
        "numpy": "np.random.rand()"
    },
    'op_compare': {
        "tolerance15": "result1 - result2 < 1e-15",
        "tolerance16": "result1 - result2 < 1e-16",
        "tolerance18": "result1 - result2 < 1e-18",
        "equal": "result1 == result2",
    }
}

all_factors = {}
for k1, v1 in factors['op_creation'].items():
    for k2, v2 in factors['op_compare'].items():
        all_factors[k1 + '_' + k2] = {
            'op_creation': v1,
            'op_compare': v2
        }

print("experiments;op_creation;op_compare;result")
for key, factor in all_factors.items():
    generated_code = template.render(factor)
    print(f"{key};{factor['op_creation']};{factor['op_compare']};", end="")
    exec(generated_code)
```

Figure 3: Template-based approach (calling)

```
import random
import argparse

def check_property(operation1, operation2, repetitions):
    correct_count = 0
    for _ in range(repetitions):
        x = random.random()
        y = random.random()
        z = random.random()

        # Dynamically evaluate operations
        result1 = eval(operation1)
        result2 = eval(operation2)

        if result1 == result2:
            correct_count += 1

    print(f"Out of {repetitions} trials, the property held {correct_count} times.")

if __name__ == "__main__":
    # Parse arguments at runtime
    parser = argparse.ArgumentParser(description="Check arithmetic property.")
    parser.add_argument('--operation1', type=str, required=True, help="First operation to compare (e.g., '(x + y) + z').")
    parser.add_argument('--operation2', type=str, required=True, help="Second operation to compare (e.g., 'x + (y + z)').")
    parser.add_argument('--repetitions', type=int, required=True, help="Number of repetitions for the check.")

    args = parser.parse_args()

    # Run the property check with runtime arguments
    check_property(args.operation1, args.operation2, args.repetitions)
```

Figure 4: CLI-based approach

```dockerfile
FROM python:3.9-slim

# Set working directorys
WORKDIR /app

# Copy Python source files
COPY generate_property_checks.py .
COPY property_template.py.jinja .

# Install Jinja
RUN pip install Jinja2
RUN pip install numpy

# Command to run Python script
CMD ["sh", "-c", "python generate_property_checks.py"]
```

(a) Docker Python

```dockerfile
FROM docker.io/gnuoctave/octave:9.2.0

WORKDIR /app
COPY . /app

RUN octave --eval 'pkg install "https://downloads.sourceforge.net/proje
CMD ["/bin/bash", "-c", "octave main.m;cat answer_associativity.txt"]
```

(b) Docker Matlab

Figure 5: Docker environments

technique, a command-line interface (CLI)-based control mechanism, and a runtime parameterization strategy. Each method offers trade-offs in terms of automation, flexibility, and scalability.

Template-Based Code Generation. This approach relies on the generation of programmatic code through templates, which allows the systematic exploration of variability factors at the source code level. Using the Jinja2 templating engine in Python, students developed templates that abstract arithmetic expressions, control structures, and experimental parameters (e.g., number of repetitions). These templates were instantiated with different configurations, producing customized program variants for each combination of factor values. Figure 2 illustrates the architecture of this approach, highlighting the parameterization points within the template. The actual invocation of template instantiation is shown in Figure 3, where a Python script dynamically generates code based on the desired factor values. This technique is especially suitable for fine-grained control and systematic testing, but requires regeneration and recompilation for each new configuration.

CLI-Based Exploration. As shown in Figure 4, the CLI-based approach exposes variability factors as runtime arguments to a fixed codebase. This method allows users to execute experiments by supplying different values directly via the command line (e.g., arithmetic operation, repetition count), enabling rapid testing without modifying the code structure.

Runtime Parameter Sweeps. The runtime parameterization technique integrates the exploration of multiple variability factors into a single execution. Rather than executing the program repeatedly with distinct arguments, all combinations of parameter values are evaluated internally using loops or Cartesian product iterations. This method, partially combined within the logic of the template-based approach in Figure 2, supports high-throughput experimentation and reduces the overhead associated with repeated compilation or launch.

It should be noted that some variability techniques are not suitable for some types of artifacts. For instance, parameterization is not ideal for changing the version of a library – a template-based approach over a Dockerfile or requirements.txt is more suited.

Containerization and environment control. To ensure reproducibility across environments and isolate system-level variability factors, students containerized their experiments using Docker. As shown in Figures 5(a) and 5(b), both Python and MATLAB-based workflows were encapsulated within Docker containers, enabling consistent execution regardless of host architecture or operating system. Containerization was further integrated with continuous integration pipelines (e.g., GitHub Actions), ensuring that the experiments were reproducible and verifiable across multiple runs and contributors.

3.4 Results' Analysis

To support a reproducible and interpretable analysis of the experimental results, the students documented their results using Jupyter notebooks. These notebooks served as the central artifact for data loading, visualization, and interpretation. After executing their experiments (through templated code, CLI invocations, or parameter sweeps), the students serialized the results into tabular formats such as CSV, with each row representing a configuration of variability factors and the corresponding computed outcome. The notebook environment allowed them to apply exploratory data analysis techniques and interpretable machine learning models, such as decision trees, to assess which variability factors (or combinations thereof) had the greatest influence on numerical divergence. By integrating code, plots, and textual discussion, the notebooks also functioned as reproducible reports, facilitating peer review and collaborative debugging.

Alongside the notebooks, students were provided with instructions that guided them through the experimental workflow, from factor modeling to result interpretation. These instructions emphasized methodological rigor, including how to isolate individual factors, explore interaction effects, and ensure consistency between execution environments. The instructions also scaffolded reproducibility practices, such as documenting configuration choices, automating executions (through CI pipelines, and encapsulating environments with Docker), to practice software engineering principles and scientific inquiry.

3.5 Students' Variability Exploration

Based on reports and discussions, the set of variability factors identified is as follows:

(1) **Random number generation**
- Range of generated numbers (e.g., `min`, `max`)
- Type of generated numbers (e.g., integer, floating-point)
- Number of decimal digits / floating-point precision
- Number of repetitions (e.g., number of samples drawn)
- Random seed (initialization of the random number generator)
- Method of generation (e.g., uniform distribution, Gaussian, specific algorithm for float generation)

(2) **Computation environment**
- Compiler used (e.g., `gcc`, `clang`, version)
- Processor / hardware architecture
- Operating System
- Programming language (e.g., C, Python, Julia)

(3) **Numerical representation and operations**
- Type of variables (e.g., `float`, `double`, `long double`)
- Number of iterations (e.g., loop count)
- Values used during computation (e.g., fixed or random operands)
- Precision of arithmetic operations (e.g., single vs. double)
- Nature of operations performed (e.g., addition, multiplication)
- Arithmetic property tested (e.g., associativity, distributivity, idempotence)

(4) **Experimental setup**
- Type of experiment conducted (e.g., testing associativity)
- Would results vary with another property? (e.g., commutativity, inverse)

Not all variability factors were explored by the students during the project. This can be attributed to several reasons. First, some factors were technically difficult to implement or instrument, such as modifying the compiler toolchain or capturing low-level floating-point behavior. Second, in some cases, students made deliberate choices to focus only on factors they found interesting or relevant to their hypothesis. Finally, the sheer number of possible variability factors made it impossible to explore all of them within the limited time and scope of the assignment. As a result, students often prioritized a subset of factors that best aligned with their experimental goals.

During the exploratory phase, a wide range of programming languages was considered, including Java, Python (in multiple versions and with or without `numpy`), C, Scratch, Matlab, Sage, JavaScript, and C++. For the actual experiments, the majority of groups (six) used Python, while one group each used Java, Matlab, JavaScript, and C. The choice of language influenced the variability factors that could be easily explored.

However, not all groups selected the same set of variability factors. The most frequently investigated were the number of repetitions and the range of generated values (`min`, `max`). Interestingly, several groups also explored the type of operation beyond associativity (e.g., testing commutativity or distributivity). This extension can be seen as a form of replication, as it diverges from the original study's focus and explores how variability behaves under different mathematical properties.

Overall, the students' work was solid, well-argued, and demonstrated thoughtful experimentation. Many projects offered interesting insights and supported their arguments with evidence drawn from systematic testing and analysis. A possible frustration is that there is no single "final truth" or universal conclusion—results can vary depending on the chosen parameters, environment, and design choices. However, it is worth noting that results tended to converge toward more consistent patterns across groups as experiments became more refined and controlled.

This kind of exploration necessarily comes with some ambiguity: part of the challenge lies not just in observing variability, but in explaining it. In many cases, offering a solid explanation would require stepping outside the scope of the course or acquiring deeper domain expertise (e.g., in compiler internals, floating-point standards, or numerical analysis). These limitations are not shortcomings, but rather reflect the complexity and richness of studying variability in computational experiments.

4 Reproducing and Replicating Research Papers

4.1 Methodological Setup

Students selected one of three instructor-curated papers. The three proposed studies were:

- **Debunking the Chessboard: Confronting GPTs Against Chess Engines to Estimate Elo Ratings and Assess Legal Move Abilities** [3] explored the capabilities of large language models (LLMs) in playing chess. Students could examine sensitivity to model versions (e.g., GPT-4o vs Claude or LLaMA), prompt engineering, sampling parameters (e.g., temperature), and the effect of using different chess engines for evaluation. We call the study "chess" for short hereafter.
- **COVID and Home Advantage in Football: An Analysis of Results and xG Data in European Leagues** [2] focused on the impact of COVID-era matches played without crowds on home advantage. Students could consider extensions to newer seasons (e.g., 2023/2024), alternative sources of xG (expected goals) data, inclusion of different leagues (including amateur or youth), and even cross-domain replications in other sports. We call the study "football" for short hereafter.
- **Programming Languages and Energy Efficiency** [21] ranked popular programming languages according to energy consumption in tasks. Students could investigate how the results varied with different programs/workloads, interpreters, compiler flags, containerization, or hardware platforms, offering rich ground for controlled replication and sensitivity analysis. We call the study "energy" for short hereafter.

These papers were chosen for their diversity in domain (LLMs, sports analytics, and energy), their reproducibility potential (open data/code), and the instructor's in-depth familiarity with the materials, enabling close guidance and nuanced discussions throughout the process. The two papers [2, 3] on chess and football were written by the instructor through technical blog posts.

The activity followed a two-phase methodology. First, each group (two students) was asked to faithfully reproduce a core subset of the original results using the provided data, tools, and methodology. This phase required setting up reproducible environments (e.g., using Docker and CI pipelines) and documenting the steps required to obtain results similar to those in the original work. Second, the students were tasked with designing and executing a replication by identifying and systematically varying factors that could affect the results, such as software versions, input data, modeling assumptions, or experimental parameters.

This setup emphasized the distinction between reproduction (same setup, same data, same environment) and replication (controlled variation of specific factors). The students were required to document the variability factors considered, justify their choice of replication strategies, and report the robustness or brittleness of the original claims in light of their findings.

4.2 Reproduction Protocol

The reproduction phase began with a thorough reading of the selected paper, emphasizing the abstract, methodology, and results sections, as well as any supplemental material or linked repository. This close analysis was essential to identify the components

required for reproduction (datasets, code, parameter settings, evaluation metrics, etc.). It can also serve to spot potential ambiguities or assumptions left implicit in the original documentation.

With this understanding, the students worked to recreate the original experimental setup as faithfully as possible. This involved acquiring the same data, reusing or adapting the original code when available, and reconstructing the computational environment using Docker. Students were encouraged to go beyond manual reproduction by automating analysis pipelines and, where appropriate, using Jupyter notebooks and tools like `nbconvert` to formalize the process. Some groups also integrated CI pipelines to automatically verify that their setups produced consistent results on all runs.

To support structured documentation and reflection, a detailed `README.md` template was provided and required for submission. This template guided students through the steps of reporting software dependencies, setting up instructions, automating scripts, and executing analyzes. It also prompted them to explicitly describe any deviations from the original study, report encountered challenges, and reflect on whether the original results could be reproduced. Rather than treating reproduction as a purely technical task, the combination of critical reading, hands-on execution, and structured documentation helped students internalize the conceptual underpinnings of reproducibility in existing research.

4.3 Replication Protocol

Where reproduction sought to re-establish the original results under identical conditions, the replication phase aimed to assess the robustness and generalizability of those results by introducing well-motivated variations. This involved modeling potential sources of variability in the original setup, ranging from input data, hardware configurations, software versions, model hyperparameters, and evaluation strategies.

Each group selected one or more variability factors to explore and justify their choices based on the assumptions or potential sensitivities of the original study. These factors were formally documented in `README.md`, using a structured table to describe their possible values and relevance. The modeling of constraints across factors (e.g., hardware-specific limitations or library versions) was also requested.

The students implemented their controlled variations and performed replication experiments, comparing the results to those obtained during the reproduction phase. Where possible, they used scripts or custom command-line interfaces (CLIs) to systematically explore different combinations of variability factors. The results were analyzed both quantitatively and interpretively, with a focus on whether the original findings remained consistent or showed signs of fragility under new conditions.

The README template again played a key role here, helping students organize and communicate their replication strategy, the experimental protocol, and the conclusions. It prompted them to provide execution instructions, analyze the results, and clearly state whether their replications supported or contradicted the original study's conclusions. This structure fostered both rigor and comparability across projects and allowed the instructor to evaluate not just technical correctness, but also scientific reasoning and reflection.

4.4 Insights and Student Contributions

The 10 groups worked on the following selections: 5 groups analyzed the football study, 3 groups examined the chess study, and 2 groups explored the energy efficiency paper.

The football study yielded the most immediate actionable insights, as detailed in the next section. The other two studies also produced relevant and high-quality work, though their nature led to more open-ended or exploratory findings.

4.4.1 Chess. For the chess study, students' work demonstrated a strong understanding of the methodological requirements of reproducible research, particularly in managing dependencies, setting up Dockerized environments, and automating analysis workflows. All groups successfully reproduced key metrics from the original study, such as Elo ratings and the frequency of illegal moves for different GPT models. Despite the reproducibility of the original results, students encountered recurring challenges that highlight broader issues in reproducible machine learning experiments. Among these were compatibility issues stemming from outdated or deprecated library methods (notably in pandas), the lack of a fully specified environment in the original study, and inconsistencies in random output due to the stochastic nature of LLMs. These challenges required thoughtful decision-making, such as pinning library versions or rewriting parts of the source code, and offered students valuable hands-on experience in debugging reproducibility problems in a real-world research context.

The replication phase provided an opportunity to explore a wide range of variability factors, including temperature, model family, prompt structure, and even chess variants such as Chess960. All groups reported that GPT-3.5-turbo-instruct was uniquely capable of playing legal and reasonably strong chess, outperforming both chat-based models and alternative architectures like LLaMA. However, attempts to generalize the study to Chess960 revealed clear limitations: models often ignored variant-specific headers and produced illegal or non-sensical moves, suggesting a strong bias toward standard chess in training data and prompting strategies. These replications confirmed the robustness of the original findings, while introducing valuable extensions and critical perspectives. Students also contributed new tools, such as parameterizable CLI-based infrastructures and refined Elo computation techniques, and provided empirical insights into LLM behavior under more diverse conditions. The work highlights both the promise and the present limitations of applying LLMs to chess, a game whose immense combinatorial complexity makes mere memorization insufficient.

A limitation acknowledged by all groups is the inherent complexity of the variability space. Each factor (such as temperature, prompt formulation, or opponent strength) can significantly impact results and requires careful, isolated manipulation to yield interpretable conclusions. The combinatorial explosion of possible configurations makes it difficult to cover the full space in a systematic way. Moreover, experimenting with LLMs often requires substantial computational resources or paid access to proprietary APIs, adding another layer of complexity and cost. Instrumenting controlled experiments can therefore be both technically and financially demanding, limiting the extent to which students or researchers can explore large portions of the variability space in practice. In this context, careful scoping of the replication effort is

essential, and the student groups demonstrated good judgment in selecting focused, tractable subsets of variability factors.

4.4.2 Energy. Two groups of students participated in the study on energy efficiency [21]. Although the topic is highly technical, it offered a concrete and reproducible benchmark with which students could engage, especially in terms of system-level experimentation, performance trade-offs, and the reproducibility of empirical software engineering results.

Both groups successfully reproduced the main results of the study, including the relative efficiency of the compiled versus interpreted languages and the broad trends observed in the language rankings. The students used Docker to ensure a controlled benchmarking environment and relied on RAPL-based energy measurement tools, as suggested by the original methodology. However, a common point of confusion stemmed from the multiple repositories and variants of the benchmarking code available online. Although the study's core contributions remain clear, the presence of forks and alternative implementations required students to carefully align their setup with the precise version of the benchmark suite used in the published results. This step, though nontrivial, offered an instructive experience in navigating research artifacts and understanding the importance of versioned and curated releases.

During the replication phase, students identified well-motivated variations, such as changing compiler flags, comparing runs on different CPU architectures, and substituting alternative implementations of the same benchmark task (e.g., different versions of a matrix multiplication algorithm). These replications generally confirmed the robustness of the original language ranking trends, but also demonstrated that fine-grained performance metrics (particularly energy consumption) could vary based on hardware configuration or implementation subtleties. Such results do not contradict the original study, but rather reinforce the need to interpret performance claims within specific environmental contexts.

The student reports also included useful contributions, such as automated CLI tools to run multiple benchmarks in batch mode, and well-structured documentation based on the provided template. In general, their work confirmed the reproducibility of the original study and provided practical extensions that highlight the nuanced nature of empirical performance evaluation. An important area identified for future work concerns the choice and coverage of the benchmark tasks themselves. Although selected programs provide a standardized and comparable testbed, the students noted that these tasks may not reflect the diversity of real-world workloads or typical language use cases. Expanding the benchmark suite to include more application-oriented or domain-specific tasks, such as data processing, machine learning, or web services, could further enrich the interpretability and generalizability of language efficiency comparisons. This insight underscores the evolving nature of benchmarking research and the value of student-led replications in questioning and extending the experimental foundations of published studies.

4.5 When students found reproducibility issues in your own work

4.5.1 Case study: Home Advantage in Football. Five groups examined a data-driven blog post written in 2021 about whether there was a measurable home-field advantage in professional football, and whether this advantage changed during the pandemic period. The analysis was motivated by the hypothesis that, due to the absence of spectators during lockdowns, the traditional home advantage would be significantly reduced. The original study collected match data from various European leagues, such as Ligue 1, La Liga, and the Premier League, and used multiple metrics to assess performance. These included both actual match results (points earned) and more refined metrics such as expected goals and expected points, which provide a probabilistic evaluation of team performance based on scoring opportunities. The data suggested that in 2020, the home-field advantage indeed diminished in certain leagues: for example, the Premier League saw 27 more points won away than at home, and in Ligue 1, there was a visible drop compared to pre-COVID seasons. In contrast, the effect was less pronounced in La Liga, raising interesting comparative questions. To evaluate significance, non-parametric statistical tests such as the Wilcoxon signed-rank test and the Mann–Whitney U test were applied, both on raw results and on expected points. Although this work was never formally published, it served as a rich foundation for a pedagogical exercise on reproducibility.

Importantly, note that the code to reproduce the results was not available, including the code to gather data and to analyze data (e.g. visualization, statistical tests). Public release of the code and data was delayed due to a gray zone about data and xPoints. Despite student requests, the nonsharing of the source code was eventually decided *on-purpose* and for pedagogical reasons. In fact, the instructor found that the original explanations were sufficient. Thus, it was an opportunity to technically challenge the students in the groups to *re-implement* the original pipeline. In a sense, it is a different form of reproducibility than chess and energy studies where code and data were available.

4.5.2 Reproducing results and discrepancies. The students were asked to reproduce the findings of the blog post. This led to a fascinating hands-on experience. Although most of the results aligned with the original observations, the students found that the p-values in the statistical tests were systematically twice as high as those reported in 2021. This discrepancy raised questions. Figure 6 specifically depicts the results of the original study for the Premier League, as well as the results of a reproduction by one of the groups. We can observe that some results are no longer statistically significant.

The first interactions with some groups lead to interesting discussions: Is it a bug on their side when applying the statistical tests? Or a wrong choice? Is it due to the extracted data? Or is it a bug in the original study? Upon investigation, the issue was traced to a change in the default behavior of the SciPy library between versions 1.6.0 (used in 2021) and 2024. The default hypothesis for statistical tests had changed from two-sided to one-sided, introducing silent variation in output without generating any errors or warnings. This underscored the fragility of computational pipelines when faced with versioning issues in scientific software.

The students also went beyond simple reproduction by engaging in replication, using alternative datasets and extending the temporal scope of the analysis. They applied the same methodology to matches from the Champions League, the Turkish Süper Lig, and post-COVID seasons (2022–2024)—allowing for new insights and validating the robustness of the observed trends.

EPL xPoints change significance with mann

	2014	2015	2016	2017	2018	2019	2020
2014							
2015	0.061728						
2016	0.207185	0.260411					
2017	0.408533	0.119547	0.291484				
2018	0.104307	0.418444	0.392634	0.208557			
2019	0.032631	0.373903	0.193936	0.077736	0.259500		
2020	0.000772	0.047791	0.015721	0.003938	0.032873	0.102317	

(a) Original

English Premier League xPoints Change Significance with Mann-Whitney U Test

	2014	2015	2016	2017	2018	2019	2020
2014	nan	nan	nan	nan	nan	nan	nan
2015	0.123055	nan	nan	nan	nan	nan	nan
2016	0.414370	0.519108	nan	nan	nan	nan	nan
2017	0.817066	0.238897	0.582968	nan	nan	nan	nan
2018	0.208613	0.835856	0.785267	0.417115	nan	nan	nan
2019	0.065263	0.748807	0.387872	0.155471	0.519001	nan	nan
2020	0.001544	0.095714	0.031442	0.007875	0.065746	0.204634	nan

(b) Reproduction

Figure 6: Comparison of results: original vs reproduction

This exercise exemplified both the challenges and the pedagogical value of reproducibility. It highlighted the importance of transparent workflows, version control, and rigorous documentation – not only to validate past results, but also to enable meaningful generalization through replication.

5 Discussion

Values and limits of "Hello world" The x+(y+z) task illustrated how simple problems can yield rich insights. Student work revealed both technical challenges (tooling, versions, etc.) and epistemological ones (what counts as reproducible?). We believe that the material (slides, lab session instructions) can be reused in other teaching contexts, especially to practice software and tools. We have experimented with similar material in a summer school for doctoral students and have received positive feedback. Another related exercise, perhaps more engaging, is as follows:

```
My banker proposed this investment to me:
You give me e ~ 2.71828...
The following year, I take 1 euro as a fee and multiply by 1.
The next year, I take 1 euro as a fee and multiply by 2.
The next year, I take 1 euro as a fee and multiply by 3.
...
After n years, I take 1 euro as a fee and multiply by n.
To retrieve my money, there is a 1 euro fee.
In 50 years, for my retirement, how much money will I have?
Write a program that answers this question
```

This exercise was used as a bonus, but the same methodology applies. The answer depends on a multiplicity of factors. Interestingly, there is a known truth through either precise computation (and control of variability factor) or analytics.

However, both exercices surfaced the real-world difficulties of reproduction: incomplete or brittle research artifacts, situations where documentation is insufficient, code does not behave as expected across environments, replication requires an unreasonable amount of effort, etc. Hence, the second part of our teaching course with real-world papers is complementary (not to say mandatory) to get the big picture of reproducibility and replicability.

Teaching variability. Looking ahead, we see an opportunity to go deeper into variability by integrating feature modeling and designing a comprehensive course module around it. This would allow for a more principled exploration of configuration spaces and interactions between variability factors. There are existing initiatives in software engineering communities to teach software product line engineering, configurable systems, and variability engineering [5]. A challenge is to adapt such content for reproducibility.

Dissemination. One open question is how to disseminate and valorize high-quality student work. Some replication projects produced insightful contributions that could be developed into co-authored publications with students. A notable point of feedback was that students now want more experience with paper reproduction integrated throughout the curriculum. This highlights the pedagogical value of engaging directly with real research.

Limits of technologies. Software technologies supporting reproducibility evolve. The choice of Docker was guided by its usage in other teaching modules, but can be overridden in the future. In fact, the course was an excuse to show the limits of existing tools. For instance, Figure 5, page 5 shows Dockerfiles that (1) do not capture versions of library; (2) rely on an ad hoc source repository to download a tool.

Instructor. We observed an ongoing tension between providing detailed instructions and allowing open-ended investigation. Pilot studies suggest that reproducibility projects can be a practical and effective teaching method [12]. Guidelines for undergraduate replication projects help scaffold research skills and foster scientific rigor [20]. Striking the right balance between "here is exactly how to do it" and "you are free, it is life" remains a key design challenge for future iterations of the course. Another limitation of our work is that there was a unique instructor, which eases seamless and consistent communication.

6 Conclusion

We reported on a self-contained reproducibility course that blends scientific thinking, software tooling, and variability analysis. We proposed a "hello world" based on floating point communication that is both simple and engaging and allows students to explore and implement variability at different levels of abstraction. Once students have mastered the basics, they can explore more advanced topics and realistic scenarios. We then challenged the students to reproduce a real-world research paper (out of three proposals) using the tools and techniques they learned and replicate it through variability analysis. Interestingly, the students were able to pinpoint potential reproducibility issues (i.e., a library version that used a different default parameter) and to propose replication (variations) of the original paper to further explore new hypotheses and test generality. We believe that part of our material can be reused (e.g., the "hello world" example). We also want to emphasize the role of variability management in the context of reproducibility. We have

taught some techniques (e.g., modeling of variability, template-based programming), but a formal, dedicated course component would allow for a more principled exploration of configuration spaces and interactions between variability factors. Our course was a way to practice software engineering and reproducibility, and also an excuse to experiment with scientific papers and the scientific methods, something not that easy to integrate into a curriculum for students not involved in research. The material of the course is available on Software Heritage [1].

As future work, we plan to repeat our course and adjust some points. First, the "hello world" example can be revisited. The example is a good starting point to introduce students to the topic, including technical aspects, but does not capture the complexity of reproducibility scenarios and real-world research artifacts. To go further, we have developed a repository of possible solutions in different languages and different variability (e.g., compiler flags) that can be confronted with students' implementations. Another direction is to use other related examples, like the "banking problems", which also involve floating-point computation issues.

Second, the idea of reproducing and then replicating papers is both engaging and useful. However, the selected papers matter a lot. An ideal scenario is to have papers that are well mastered by instructors, in such a way the communication between instructors and students is efficient and the paper can be reproduced with high confidence. Another advantage is to know beforehand if the paper is likely to be replicated or not, avoiding too technical or heavy works for students. Finally, the replication of the article can be used as a way to further explore new hypotheses and test the generality of the original article. It is actually the most exciting direction, as both instructors and students are deeply engaged with the paper, and the paper can be replicated with high confidence. From this regard, we might co-author a paper with students on the new results obtained during the course – not about reproducibility per se, but about home advantage in football.

Acknowledgments. We thank Fraida Fund for early discussions and sharing her experience in teaching reproducibility. The authors thank Paul Temple for discussions, presentations, and work on this topic. The authors thank the 20 students at INSA Rennes for their active participation throughout the course and beyond. This research is funded by Inria RIPOST-EA.

References

[1] Mathieu Acher. 2025. REP-INSA2425. https://archive.softwareheritage.org/swh:1:dir:89ad4e83c94a17411027ee891c9cb297a4df02c5 Archived in Software Heritage.

[2] Mathieu Acher. 2021. COVID and Home Advantage in Football: An Analysis of Results and xG Data in European Leagues. https://blog.mathieuacher.com/FootballAnalysis-xG-COVIDHome/.

[3] Mathieu Acher. 2023. Debunking the Chessboard: Confronting GPTs Against Chess Engines to Estimate Elo Ratings and Assess Legal Move Abilities. https://blog.mathieuacher.com/GPTsChessEloRatingLegalMoves/.

[4] Mathieu Acher, Benoît Combemale, Georges Aaron Randrianaina, and Jean-Marc Jézéquel. 2024. Embracing Deep Variability For Reproducibility and Replicability. In *Proceedings of the 2nd ACM Conference on Reproducibility and Replicability, ACM REP 2024, Rennes, France, June 18-20, 2024*. ACM. doi:10.1145/3641525.3663621

[5] Mathieu Acher, Roberto E. Lopez-Herrejon, and Rick Rabiser. 2017. Teaching Software Product Lines: A Snapshot of Current Practices and Challenges. *ACM Trans. Comput. Educ.* 18, 1 (2017), 2:1–2:31. doi:10.1145/3088440

[6] Thomas Cokelaer, Sarah Cohen-Boulakia, and Frédéric Lemoine. 2023. Reprohackathons: promoting reproducibility in bioinformatics through training. *Bioinformatics* 39, Supplement_1 (June 2023).

[7] R. Döscher, M. Acosta, A. Alessandri, P. Anthoni, T. Arsouze, T. Bergman, R. Bernardello, S. Boussetta, L.-P. Caron, G. Carver, M. Castrillo, F. Catalano, I. Cvijanovic, P. Davini, E. Dekker, F. J. Doblas-Reyes, D. Docquier, P. Echevarria, U. Fladrich, R. Fuentes-Franco, M. Gröger, J. v. Hardenberg, J. Hieronymus, M. P. Karami, J.-P. Keskinen, T. Koenigk, R. Makkonen, F. Massonnet, M. Ménégoz, P. A. Miller, E. Moreno-Chamarro, L. Nieradzik, T. van Noije, P. Nolan, D. O'Donnell, P. Ollinaho, G. van den Oord, P. Ortega, O. T. Prims, A. Ramos, T. Reerink, C. Rousset, Y. Ruprich-Robert, P. Le Sager, T. Schmith, R. Schrödner, F. Serva, V. Sicardi, M. Sloth Madsen, B. Smith, T. Tian, E. Tourigny, P. Uotila, M. Vancoppenolle, S. Wang, D. Wårlind, U. Willén, K. Wyser, S. Yang, X. Yepes-Arbós, and Q. Zhang. 2022. The EC-Earth3 Earth system model for the Coupled Model Intercomparison Project 6. *Geoscientific Model Development* 15, 7 (2022), 2973–3020. doi:10.5194/gmd-15-2973-2022

[8] Steve M Easterbrook and Timothy C Johns. 2009. Engineering the software for understanding climate change. *Computing in science & engineering* 11, 6 (2009), 65–74.

[9] Fraida Fund. 2023. We Need More Reproducibility Content Across the Computer Science Curriculum. In *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability (ACM REP '23)*. Association for Computing Machinery, New York, NY, USA, 97–101. doi:10.1145/3589806.3600033

[10] Tristan Glatard, Lindsay B Lewis, Rafael Ferreira da Silva, Reza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Rousseau, Tarek Sherif, Ewa Deelman, et al. 2015. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in neuroinformatics* 9 (2015), 12.

[11] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[12] Nestoras Karathanasis, Daniel Hwang, Vibol Heng, Rimal Abhimannyu, Phillip Slogoff-Sevilla, Gina Buchel, Victoria Frisbie, Peiyao Li, Dafni Kryoneriti, and Isidore Rigoutsos. 2022. Reproducibility efforts as a teaching tool: A pilot study. *PLOS Computational Biology* 18, 11 (2022). doi:10.1371/journal.pcbi.1010957

[13] Gregory Kiar, Jeanette A. Mumford, Ting Xu, Joshua T. Vogelstein, Tristan Glatard, and Michael P. Milham. 2024. Why experimental variation in neuroimaging should be embraced. *Nature Communications* 15, 1 (31 Oct 2024), 9411. doi:10.1038/s41467-024-53743-y

[14] Christopher G Knight, Sylvia HE Knight, Neil Massey, Tolu Aina, Carl Christensen, Dave J Frame, Jamie A Kettleborough, Andrew Martin, Stephen Pascoe, Ben Sanderson, et al. 2007. Association of parameter, software, and hardware variation with large-scale behavior across 57,000 climate models. *Proceedings of the National Academy of Sciences* 104, 30 (2007), 12259–12264.

[15] Ana Lucic, Maurits Bleeker, Sami Jullien, Samarth Bhargav, and Maarten de Rijke. 2022. Reproducibility as a Mechanism for Teaching Fairness, Accountability, Confidentiality, and Transparency in Artificial Intelligence. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36.

[16] François Massonnet, Martin Ménégoz, Mario Acosta, Xavier Yepes-Arbós, Eleftheria Exarchou, and Francisco J Doblas-Reyes. 2020. Replicability of the EC-Earth3 Earth system model under a change in computing environment. *Geoscientific Model Development* 13, 3 (2020), 1165–1178.

[17] Wolfgang Mauerer, Stefan Klessinger, and Stefanie Scherzinger. 2023. Beyond the Badge: Reproducibility Engineering as a Lifetime Skill. In *Proceedings of the 4th International Workshop on Software Engineering Education for the Next Generation (SEENG '22)*.

[18] Olivier Mesnard and Lorena A Barba. 2017. Reproducible and replicable computational fluid dynamics: it's harder than you think. *Computing in Science & Engineering* 19, 4 (2017), 44–55.

[19] K. Jarrod Millman, Matthew Brett, Ross Barnowski, and Jean-Baptiste Poline. 2018. Teaching computational reproducibility for neuroimaging. *Frontiers in Neuroscience* 12 (2018).

[20] David Moreau and Kristina Wiebels. 2023. Ten simple rules for designing and conducting undergraduate replication projects. *PLOS Computational Biology* 19, 3 (2023), e1010957. doi:10.1371/journal.pcbi.1010957

[21] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2021. Ranking programming languages by energy efficiency. *Science of Computer Programming* 205 (2021), 102609. doi:10.1016/j.scico.2021.102609

[22] Lars Vilhuber, Hyuk Harry Son, Meredith Welch, David N. Wasser, and Michael Darisse. 2022. Teaching for Large-Scale Reproducibility Verification. *Journal of Statistics and Data Science Education* 30, 3 (2022).

[23] Yibo Wang, Ying Wang, Tingwei Zhang, Yue Yu, Shing-Chi Cheung, Hai Yu, and Zhiliang Zhu. 2023. Can Machine Learning Pipelines Be Better Configured?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. Association for Computing Machinery, 463–475. doi:10.1145/3611643.3616352

[24] Burak Yildiz, Hayley Hung, Jesse H. Krijthe, Cynthia C.S. Liem, Marco Loog, Gosia Migut, Frans A. Oliehoek, Annibale Panichella, Przemysław Pawełczak, Stjepan Picek, et al. 2021. ReproducedPapers.org: Openly teaching and structuring machine learning reproducibility. In *Reproducible Research in Pattern Recognition: Third International Workshop, RRPR 2021*.