CS310 Project Specification:
Self-proving machine learning models with per-input verification

Howard Cheung
5514611

Supervised by Dr. Matthias C. Caro

October 2025

1 Problem statement

1.1 Motivation

Modern machine learning models and Large Language Models (LLMs) achieve outstanding *average* accuracy across benchmarks [Artificial Analysis, 2025], providing empirical evidence that they are *usually* right. However, they offer no formal guarantee that any *particular* output is correct for a specific input, due to the risk of subtle errors or "hallucinations" [Huang et al., 2025]. This poses a serious reliability gap in high-stakes domains such as healthcare, finance, mathematics, and law, where even a single incorrect answer can have severe consequences.

In this work we ask the following question:

Can a model provide not only an answer but also a formal proof that the answer is correct for the specific input at hand?

If we can solve this, we move from "usually right" to "provably right". We no longer have to bet on probabilities when we cannot afford to take any chances, and trustworthy AI could be deployed in more critical fields.

1.2 Prior work

Many strategies have been proposed to verify the reasoning of LLMs. One approach is to prompt models to explain their reasoning in natural language; however, the model may instead attempt to persuade the user of an incorrect answer [Turpin et al., 2023]. Another is to ask the LLM to defend its responses, but even when the answer is correct, it may still fail to produce a logically convincing proof [Wang et al., 2023]. Using chain-of-thought (CoT) prompting to reveal intermediate reasoning steps [Wei et al., 2022] can improve interpretability and sometimes accuracy, as demonstrated in many mainstream reasoning LLMs, such as `o1` [OpenAI, 2024], `Claude Sonnet 4.5` [Anthropic, 2025], and `Gemini 2.5 Pro` [Google DeepMind, 2025]. Even so, subsequent studies have shown that CoT reasoning *itself* can be fabricated or internally inconsistent, producing justifications that sound plausible but are not genuinely correct [Cheng et al., 2025]. Ultimately, these approaches may improve user trust or interpretability, but none can formally prove correctness *on a per-input basis*.

1.3 The self-proving framework

A recent framework proposed by Amit et al. [2024] introduces the concept of **self-proving models** to address this lack of per-input verification using interactive proofs [Goldwasser et al., 1985]. Instead of merely generating an output y from an input x, a self-proving model P_θ (the prover) produces an interactive proof transcript π alongside the output y through a series of queries and responses with a fixed verifier V; the transcript π is then checked by V, which accepts only if it is convinced that the answer y is correct and rejects otherwise[1].

1.4 Gap and project direction

However, the study by Amit et al. [2024] primarily served as a *proof of concept*, focusing on a simple arithmetic task—the greatest common divisor (GCD) problem using Bézout's identity—and left open the broader question of how these ideas might generalise to more complex reasoning tasks.

The notion of per-input verification could, in principle, be extended to language-based tasks like legal or policy summarisation, where both correctness and comprehensive coverage matter. However, unlike mathematics, correctness in such domains is often subjective and context-dependent, making formal verification far less straightforward. If mathematical problems can be generalised robustly, this line of work could eventually inspire verification methods for more ambiguous settings, but that remains an extension beyond the scope of this project.

Our project therefore begins with **mathematical** domains where correctness can be defined unambiguously. In particular, we apply the self-proving framework to areas of mathematics beyond number theory, providing both provable guarantees and implementations of self-proving models for at least one new mathematical problem. Moreover, this opens up some interesting possibilities for evaluation of the results, which will be elaborated in Section 3.3.

2 Objectives

The overarching aim is to extend the self-proving models framework beyond its original toy setting (GCD) and explore whether such per-input verification can hold for harder mathematical problems. The objectives can be divided into two categories:

- theoretical: developing the mathematical foundations and formal understanding of the framework;
- practical: implementing the theory in code and producing a functional demonstration.

2.1 Theoretical objectives

(1) Familiarisation with literature and foundation building.

- **Description:** Develop a thorough understanding of the mathematical and formal knowledge of the self-proving framework introduced by Amit et al. [2024]. This includes studying the theory of *interactive proofs*, formal definitions of *verifiers* and *provers*, and the

[1] More precisely, any incorrect y passes only with negligible probability.

three training methods—*Transcript Learning* (TL), *Reinforcement Learning from Verifier Feedback* (RLVF), and *Annotated Transcript Learning* (ATL). Supplementary resources like the talk on YouTube will be utilised.

- **Deliverable:** Study notes (typed in `Markdown` using Obsidian) summarising key definitions, intuition behind the theory, and relationships between core concepts. These notes will serve both as reference material for supervisor meetings and as a foundation for the Literature Review in the final report.

② Extension to a more complex mathematical task.

- **Description:** Select and define a new mathematical problem to apply the self-proving framework beyond the GCD example. One potential candidate is *Carathéodory's Theorem* for convex set containment, which states that any point inside a convex hull in $\mathbb{R}^d$ can be expressed as a convex combination of at most $d+1$ points. This property naturally lends itself to a prover-verifier setup, where the prover must supply such a combination and the verifier checks its validity efficiently. The goal is to determine how correctness and verifiability can be clearly defined for this problem while keeping it simple enough to implement in the later demo.
- **Deliverable:** A short write-up summarising the chosen problem, its formal definition, and an outline of how a prover-verifier setup could be built for it. This will be discussed with the supervisor to assess feasibility and guide the practical stage.

③ Exploratory work.

- **Description:** Reflect on insights gained from applying the framework to the new mathematical problem and explore whether similar principles could extend to other domains like combinatorics (e.g. graph-colourability or connectivity proofs), or linear algebra (e.g. matrix inversion verification). These would be pursued only in a best-case scenario as exploratory directions once the primary mathematical task has been completed and they involve identifying structural patterns that make a task suitable for self-proving models. If time permits, connections to quantum-verifiable models could be explored with the supervisor, but this is a stretch goal that will only happen if the main objectives are completed smoothly ahead of schedule.
- **Deliverable:** To be defined in consultation with the supervisor. Possible outcomes may include a short conceptual note or meeting summary that outlines promising directions to be refined into follow-up objectives on new problems or a generalisation.

2.2 Practical objectives

① Reproduction of the baseline system.

- **Description:** Fork the official repository by Amit et al. [2024] to set up a local development environment. Reproduce the original GCD-based experiment using the same framework built using nanoGPT. The goal is to understand the code structure, verify that the results match those reported in the paper, and gain practical experience with the prover-verifier interaction, implementation of proof generation, training configuration, and evaluation metrics. Once the baseline runs smoothly, I plan to experiment

with scaling the setup to larger open-source/weight models (such as GPT-OSS) to see how the framework behaves at a bigger scale.

- **Deliverable:**
  - A forked and functioning GitHub repository successfully replicating the GCD experiment.
  - Notes summarising the repository structure, key functions (proof generation and verification), system behaviour, and training results, along with thoughts on extending the setup to larger models.

② Implementation of the new problem.

- **Description:** Implement and train self-proving models for the new mathematical problem defined earlier. This will include programming a verifier algorithm, preparing the datasets, and applying the training methods described by Amit et al. [2024]: TL, RLVF, and ATL.
- **Deliverable:** A functional implementation of the self-proving framework applied to the chosen mathematical problem in a GitHub repository, with clear documentation explaining the verifier design, dataset setup, and training outcomes, which will serve as the basis for evaluation and analysis in the next stage of the project.

③ Evaluation and analysis.

- **Description:** Evaluate the implemented system by comparing correctness and verifiability between the baseline (GCD) and the new mathematical task quantitatively. Analyse failure cases, model behaviour, and scalability to assess the strengths and limitations of the framework.
- **Deliverable:** A document summarising experimental results, key metrics, and insights into model performance and generalisability, which will be included in the progress and final report.

As this is a new and fast-moving area of research, the precise scope and direction are expected to develop over time. The objectives above mark the early and tangible milestones.

3 Methods & methodology

3.1 Development methodology

This project will adopt an iterative workflow. Each iteration has three stages:

1. Define or refine a theoretical component (e.g. verifier design or training objective).
2. Implement and run experiments to test it.
3. Analyse outcomes and update the plan based on results.

Findings and milestones will be reviewed during fortnightly supervision meetings, so subsequent work builds on validated results and insights.

As mentioned, the project will start by reproducing the baseline GCD experiment from Amit et al. [2024] using their repository. If the fork integrates cleanly, further work will build on it; otherwise (e.g. if there are dependency issues, limited modularity, or incompatibility with any extensions), a standalone implementation will be developed.

3.2 Data generation

The project will generate all training data programmatically rather than relying on external datasets. Each sample represents an interaction between a *prover* (the model) and a *verifier* (a deterministic checking algorithm).

For a given input x, the ground-truth output $F^*(x)$ is computed using a trusted algorithm, and a valid proof transcript π is produced by the model. The resulting triplets (x, y, π) form the training data for TL. In ATL, intermediate reasoning steps are inserted as annotations within π. In RLVF, data are generated online as the model interacts directly with the verifier and receives binary accept/reject feedback.

All datasets will be created with fixed random seeds and clear documentation of the generation parameters, so experiments can be reproduced.

3.3 Analysis and evaluation

The evaluation strategy largely follows the framework introduced by Amit et al. [2024], with additional analyses specific to this project. Model performance will be assessed quantitatively along two core dimensions:

- **Correctness:** the proportion of model outputs y matching the ground-truth $F^*(x)$.
- **Verifiability:** the proportion of outputs whose proofs π are accepted by the verifier V.

Results will be reported as averages over multiple seeds with standard deviations and plotted as learning curves showing correctness and verifiability over training iterations.

In addition to the standard evaluation, three complementary analyses will be performed to gain a deeper understanding of model behaviour:

- **Sample efficiency:** how quickly each training method improves verifiability with increasing data or training steps, indicating how efficiently the model learns from verifier feedback.
- **Proof-level statistics:** examination of typical proof length, structure, and interaction complexity (e.g. number of rounds), which are expected to vary depending on the mathematical task and verifier design. This helps identify how the complexity of the task influences the difficulty of achieving high verifiability percentage.
- **Cross-task comparison:** comparison between the baseline GCD experiment and the new mathematical task to evaluate whether the three training methods remain effective as proof structures and verifier complexity change, and how well the self-proving framework generalises to different problem structures.

3.4 Version control and transparency

All code, notes, experiment configurations, and results (including metrics, visualisations, and summaries) will be version-controlled using `Git` and hosted on a GitHub repository accessible to the supervisor, with branches representing major iterations.

4 Timetable

Time	Focus/work	Official deadline
* * * * * **Start of Term 1** * * * * *		
Week 1 Oct 6 – Oct 10	Project initiation; kick-off meeting with the supervisor.	—
Week 2 Oct 13 – Oct 17	Write-up and submission of this document (**Project Specification**).	**Project Spec.** Thursday, Oct 16
Week 3 Oct 20 – Oct 24	Literature review on interactive proofs, provers/verifiers, TL/RLVF/ATL; produce `markdown` notes.	—
Week 4 Oct 27 – Oct 31	Reproduce GCD baseline experiment from Amit et al. [2024]. Fork the repo, set up the development environment, install dependencies, and study the code structure, focusing on data generation and verifier implementation.	—
Week 5 Nov 3 – Nov 7	Complete the baseline reproduction; execute the full training and evaluation scripts, debugging any runtime issues. Check metrics match paper; prepare short internal summary.	—
Week 6 Nov 10 – Nov 14	Choose a mathematical problem suitable for self-proving setup; formalise its definition and correctness condition[2]. Draft pseudocode for both prover's expected output and verifier's checking algorithm.	—
Week 7 Nov 17 – Nov 21	Implement the verifier algorithm and dataset generation scripts for the new mathematical task.	—
Week 8 Nov 24 – Nov 28	Train the self-proving model on the new mathematical task using TL, ATL, and RLVF.	—

[2] just like Bézout's identity for the GCD problem

Time	Focus/work	Official deadline
Week 9 Dec 1 – Dec 5	Evaluate the model's performance on the new task; measure correctness and verifiability, and analyse initial results using the three metrics from Section 3.3.	—
Week 10 Dec 8 – Dec 12	Clean repo, summarise results, and prepare short internal write-up for **Progress Report**.	—
* * * * * **End of Term 1** * * * * *		
Christmas break Dec 13 – Jan 11	Buffer time; start first draft of the **Progress Report**.	—
* * * * * **Start of Term 2** * * * * *		
Week 1 Jan 12 – Jan 16	Discuss progress and next steps with supervisor. Prepare and review the first draft of the **Progress Report**.	—
Week 2 Jan 19 – Jan 23	Finalise and submit the **Progress Report**.	**Progress Report** Thursday, Jan 22
Week 3 Jan 26 – Jan 30	*Exploratory work phase.* Possible directions include: (1) extending the framework to additional mathematical problems identified in Term 1; (2) generalising findings and developing theoretical insights about which classes of mathematical problems are naturally suited for self-proving models. Specific objectives will be defined based on Term 1 findings and **Progress Report** discussions.	—
Week 4 Feb 2 – Feb 6		—
Week 5 Feb 9 – Feb 13		—
Week 6 Feb 16 – Feb 20		—
Week 7 Feb 23 – Feb 27		—
Week 8 Mar 2 – Mar 6	Consolidate all experimental (and/or theoretical) results. Finalise code, make sure it is well-documented and the repo is clean. Prepare a summary of key findings and visualisations for the **Final Report**.	—
Week 9 Mar 9 – Mar 13	Make the *Final Report Plan*[3], outlining the structure, key arguments, and results to be included.	*Final Report Plan* Friday, Mar 13

[3]This *might* be optional, but it is recommended for additional feedback.

Time	Focus/work	Official deadline
Week 10 Mar 16 – Mar 20	Meet with supervisor to review the *Final Report Plan* and discuss the scope of the **Project Management Report** before the Easter break.	—
* * * * * **End of Term 2** * * * * *		
Easter break Mar 21 – Apr 26	Finish the write-up of the **Project Management Report** and the **Final Report**.	**PM Report** & **Final Report** Thursday, Apr 9
* * * * * **Start of Term 3** * * * * *		
Early weeks Apr 27 onwards Week 2 – 3	Prepare for the **Viva**: create presentation slides, prepare demo, and review all project work and key literature for the Q&A session.	**Viva** Monday, May 4 – *Friday, May 14*[4]

5 Resources & risks

5.1 Software

The software stack will consist of free tools.

- The programming language of choice is `Python` for its readability, extensive machine-learning ecosystem, and native support for GPU acceleration via PyTorch.
- Version control will be managed using `Git` and GitHub.
- Obsidian will be used for maintaining notes and meeting summaries.
- Overleaf/LaTeX will be used for preparing official reports.
- Development will be carried out in VSCode.

All datasets are generated programmatically, so no external data is required. Key references, including Amit et al. [2024] and supporting literature on interactive proofs and verifier-feedback training, are downloaded for local access.

5.2 Hardware

The project requires computation for training and verifying small-to-medium-scale ML models.

Development and light experimentation will be conducted on a MacBook Pro (M2 Pro, 32GB RAM, 1TB storage), which provides sufficient performance for small-scale prototyping.

For heavier computation, the project will make use of the DCS compute infrastructure, which includes both Compute Nodes and the Batch Compute System:

[4]Exact date TBC. At the time of writing, the website says `Friday 14th May 2026`, but 14th May 2026 is a Thursday. It is therefore unclear whether the intended date is *Thursday 14th* or *Friday 15th May 2026*.

- Compute Nodes: e.g. `stone-01 - 04` (GTX 1650 4 GB), `cobra-01 - 02` (RTX 3050 8 GB), `panda-01 - 02` (GTX 1080 8 GB). These are suitable for mid-sized GPU workloads and debugging GPU-dependent code.
- Batch Compute System: uses the SLURM scheduler, providing access to high-end nodes such as `gecko` (A10 GPUs 24 GB) and `falcon` (A5000 GPUs 24 GB), as well as `tiger` CPU partitions with 64-core EPYC processors. This system is designed for long-running or resource-intensive jobs.

Both are accessible to third-year students and support PyTorch workloads.

Note. If the Batch Compute System is used, then the following acknowledgement will be included as per the request on the webpage:

"The authors acknowledge the use of the Batch Compute System in the Department of Computer Science at the University of Warwick, and associated support services, in the completion of this work."

Risk	**Impact**	**Mitigation strategy**
Hardware failure or device loss	Medium	All project materials are backed up daily to the private GitHub repository whenever changes are made. In case of device failure, work can continue on the DCS machines and the compute nodes or the batch compute system if necessary.
Limited access to high-performance compute resources	Medium	If the DCS GPU queues are temporarily full, jobs will be rescheduled or left in the queue. If waiting times grow too long, cloud-based alternatives like Google Colab will be used instead. Running lighter prototype versions locally on the MacBook will serve as a fallback option.
Dependency conflicts or library version changes	Low	A dedicated Python virtual environment and a `requirements.txt` file will be maintained to keep dependencies consistent across machines.
Scope creep or time overrun	High	Progress and priorities will be reviewed fortnightly with the supervisor. The timetable is optimistic, with the winter break reserved for delays.

6 Legal, social, ethical and professional issues

This project does not raise any direct legal, social, or ethical concerns. All training data is generated programmatically—since no external datasets or human participants are used, there are no privacy issues.

From a broader ethical perspective, this project *contributes positively* to AI safety by developing models that can verify the correctness of their own outputs, aligning with efforts to make AI systems more reliable and transparent.

7 References

Artificial Analysis [2025]. *LLM Leaderboard: Comparison of Over 100 Models*. Accessed: 2025-10-10. URL: https://artificialanalysis.ai/leaderboards/models.

Huang, Lei, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu [2025]. "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions". In: *ACM Transactions on Information Systems* 43.2. ISSN: 1558-2868. DOI: 10.1145/3703155. URL: https://doi.org/10.1145/3703155.

Turpin, Miles, Julian Michael, Ethan Perez, and Samuel R. Bowman [2023]. "Language models don't always say what they think: unfaithful explanations in chain-of-thought prompting". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc.

Wang, Boshi, Xiang Yue, and Huan Sun [Dec. 2023]. "Can ChatGPT Defend its Belief in Truth? Evaluating LLM Reasoning via Debate". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 11865–11881. DOI: 10.18653/v1/2023.findings-emnlp.795. URL: https://aclanthology.org/2023.findings-emnlp.795/.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter brian, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou [2022]. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 24824–24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

OpenAI [2024]. *OpenAI o1 System Card*. arXiv: 2412.16720 [cs.AI]. URL: https://arxiv.org/abs/2412.16720.

Anthropic [Sept. 2025]. *Claude Sonnet 4.5: Announcement and System Card*. System card: https://www.anthropic.com/claude-sonnet-4-5-system-card. Accessed: 2025-10-14. URL: https://www.anthropic.com/news/claude-sonnet-4-5.

Google DeepMind [2025]. *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities*. Tech. rep. Accessed: 2025-10-14. Google DeepMind. URL: https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf.

Cheng, Jiahao, Tiancheng Su, Jia Yuan, Guoxiu He, Jiawei Liu, Xinqi Tao, Jingwen Xie, and Huaxia Li [2025]. "Chain-of-Thought Prompting Obscures Hallucination Cues in Large Language Models: An Empirical Evaluation". In: *arXiv preprint arXiv:2506.17088v3*. Accepted to EMNLP 2025 Findings (to appear). URL: https://arxiv.org/abs/2506.17088v3.

Amit, Noga, Shafi Goldwasser, Orr Paradise, and Guy N. Rothblum [2024]. "Models That Prove Their Own Correctness". In: *ICML 2024 Workshop on Theoretical Foundations of Foundation Models*. URL: https://openreview.net/forum?id=yCm00lPmBo.

Goldwasser, S, S Micali, and C Rackoff [1985]. "The knowledge complexity of interactive proof-systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, pp. 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178. URL: https://doi.org/10.1145/22145.22178.