CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Autoformalization for Agda via Fine-tuning Large Language Models

Master's thesis in Computer science and engineering

Pei Huang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Autoformalization for Agda via Fine-tuning Large Language Models

Pei Huang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Autoformalization for Agda via Fine-tuning Large Language Models

Pei Huang

Supervisor: Aarne Ranta, Department of Computer Science and Engineering
Examiner: Thierry Coquand, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2025

Autoformalization for Agda via Fine-tuning Large Language Models

Pei Huang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Abstract text about your project in Computer Science and Engineering.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

Here, you can say thank you to your supervisor(s), company advisors and other people that supported you during your project.

Name Familyname, Gothenburg, 2025-05-22

Contents

List of Figures

List of Tables

1

Introduction

Autoformalization is the task of translating mathematical natural language expressions into fully formalized, machine-verifiable forms. Recently, many attempts at autoformalization are based on training or fine-tuning large language models. However, these related works have encountered a key problem: it is very difficult to build or even find a high-quality, large parallel corpus of formal language and natural language. This difficulty is faced by all formal languages, not only for the most popular Lean, not to mention Agda, which has relatively fewer codes. In the work of Jiang et al. [1], they used a large language model, the GPT-4, to convert the formal language expressions of Isabelle and Lean into English to constructe a dataset containing 332K informal-formal pairs. Compared to using well-trained mathematics and computer experts to perform manual translation to build a dataset, this method is quite cheap and convenient, saving a lot of time and money. However, the disadvantages are also difficult to ignore. First, the accuracy of the translation results based on the GPT-4 is not optimistic. Jiang et al. estimated that the accuracy of the MMA dataset they constructed was about 74% based on sampling, which is obviously insufficient for further fine-tuning the large language model. Noise or even incorrect translation may cause the model to learn incorrect translation patterns, thereby affecting the overall performance of the model. In addition, the risk of hallucinations in the data output by large language models cannot be ignored. Their subsequent experimental results also verified these shortcomings well. The large language model fine-tuned on MMA only produced 29-31% of acceptable sentences.

Different from the method of constructing MMA, though *informath* project, this paper uses the collected Dedukti formal language expressions from different sources as an intermediary to convert them into formal languages Agda, Coq, and Lean expressions, and generate diverse and controlled natural language expressions in three natural languages: English, French, and Swedish, and constructing a dataset (Synthetic Multilanguage Autoformalization Dataset, SMAD) containing approximately 30K informal-formal pairs. This means that each group of data in the dataset consists of four formal languages with the same meaning and different expressions in three corresponding natural languages (4 to N). Most importantly, with the help of *Informath*, we can ensure that each informal-formal pair in the dataset is fully correct, which can improve the quality of subsequent fine-tuning of LLMs.

In this work, we fine-tuned the open-source large language model Qwen2.5-7B[] on SMAD and established an autoformalization system for Agda for the first time. To

ensure the reliability and generalization of model testing, we used the Dedukti code from different sources to the training set data to construct the test set. The Blue-4 Score of the fine-tuned model on the test set reached 76.16, which was 48.75% higher than the score of 51.20 of the unfine-tuned baseline model. Other metrics including ROUGE-1/2/L also improved by 18.57%/40.44%/38.43% respectively compared with the baseline model. At the same time, we also checked the syntax correctness of the Agda code generated by the model. Compared with the error rate of the baseline model as high as 83.76%, the syntax error rate of the fine-tuned model dropped sharply to less than 8%. In addition, we also verified whether joint training of multiple formal languages and/or multiple natural languages can help improve the quality or training efficiency of autoformalization of Agda, a formal language with a shortage of code. Experimental results show that multi-formal/multi-natural language joint training can effectively improve various indicators of the model when the number of Agda-Eng samples is small, among which the reduction of syntax error is the most obvious.

This work differs from MMA in four key ways:

- Data source: We use the Informath pipeline (GF + Dedukti) to generate formal $\leftrightarrow$ natural pairs with controlled paraphrase diversity, rather than LLM-synthesized informal text.
- Formal systems: We include Agda for the first time in autoformalization experiments, alongside Coq, Lean, and Dedukti.
- Natural languages: We extend to three human languages (English, French, Swedish), leveraging GF's multilingual grammars.
- Evaluation bias: Our test set and training set come from the same source, both generated by the Informath project. This bias partly explains our very high BLEU-4 and low syntax-error rates, which we analyze in Section 5.

Our contributions are:

- A technical study of LLM fine-tuning (LoRA on Qwen2.5-7B) for Agda autoformalization.
- A systematic comparison of single-language vs. joint-language training (for formal and natural languages separately).
- A study of the impact of the proportion of Agda code on joint training.
- An analysis of dataset bias and additional evaluations on genuinely natural English statements for the Top 100 theorems.

1.1 Datasets

Table 1.1 presents a comparison of how the proposition `even 0` is represented across different formal systems (Dedukti, Agda, Coq, Lean) and its corresponding informal mathematical descriptions in English, French, and Swedish. These examples are

part of the SMAD dataset, which aims to bridge the gap between formal proofs and natural mathematical language.

Table 1.1: Examples of SMAD

Language	Expression(s)
Dedukti	`prop10 : Proof (even 0) .`
Agda	`postulate prop10 : even 0`
Coq	`Axiom prop10 : even 0 .`
Lean	`axiom prop10 : even 0`
English	Prop10. We can prove that 0 is even. / Prop10. 0 is even.
French	Prop10. Nous pouvons démontrer que 0 est pair. / Prop10. 0 est pair.
Swedish	Prop10. Vi kan bevisa att 0 är jämnt. / Prop10. 0 är jämnt.

1.2 Problem Statement

This paper aims to build an autoformalization system for Agda by fine-tuning LLMs. At the same time, we also aim to verify whether joint training of multiple formal languages can help improve the quality or training efficiency of LLMs in translating Agda. Similarly, does joint training of multiple natural languages help? For this purpose, we build a parallel dataset (SMAD) containing 4 formal languages and 3 natural languages. Our dataset is different from MMA in that it does not come from text generated by LLMs, but relies on multi-language generation from the GF-based *Informath* project. Different slicing strategies on SMAD allow us to conduct experiments such as "all formals languages - English" vs. "Agda - English" (multiple formal languages vs. single formal language) and "Agda - all naturals languages" vs. "Agda - English" (multiple natural languages vs. single natural language). We also study the impact of the scenario of scarce Agda code resources on the fine-tuned model by downsampling the Agda-English data. This can also verify whether adding "other formal language-natural language pairs" can improve the Agda autoformalization ability of the fine-tuned model. Finally, we also tested the informalization ability of the fine-tuned model, that is, input formal statements into the model and check whether the output natural language is consistent with the original meaning and whether it is fluent and natural.

2

Theory

In this chapter, we will introduce the relevant technical details involved in the paper.

2.1 Large Language Models

Large language models are deep neural networks typically built on the Transformer architecture and trained with self-supervised objectives on massive corpora of text and code. During pre-training, they learn the joint probability distributions of words and sentences, thereby acquiring rich representations of syntax, semantics, and world knowledge. These representations can then be adapted to downstream tasks such as translation, question answering, or code generation. Key features of the Transformer include:

- Self-Attention: Each token attends to every other token in the input sequence by computing pairwise attention weights, allowing the model to capture long-range dependencies efficiently.
- Parallelism and Scalability: Unlike recurrent architectures, Transformers process all tokens simultaneously and can be scaled by stacking dozens or even hundreds of identical Transformer blocks, enabling deep composition of representations.

LLMs have demonstrated success in code-related tasks (e.g., Codex) and even in generating formal proofs (e.g., GPT-f). However, out-of-the-box LLMs often struggle with the rigid syntax and semantics of proof assistants, necessitating domain-specific fine-tuning on parallel datasets of natural language and formal code.

2.2 Qwen2.5-7B-Instruct

Qwen2.5-7B-Instruct is an open-source, 7 billion-parameter multilingual causal language model developed by Alibaba's Qwen team. This instruction-tuned variant builds on the base Qwen-2.5B model and has the following features:

- Model Type: Causal Language Models
- Training Stage: Pretraining and Post-training

- Architecture Enhancements: Rotary positional embeddings (RoPE), SwiGLU activations, RMSNorm layers, and Attention QKV bias
- Number of Parameters: 7.61 billion total; 6.53 billion non-embedding
- Number of Layers: 28
- Number of Attention Heads (GQA): 28 for Q and 4 for KV
- Context Window: Full 131,072 tokens and up to 8,192 tokens for generation

We selected Qwen2.5-7B-Instruct for our experiments because:

- Balanced Scale: At 7 billion parameters, it strikes a practical balance between functionality and efficiency, suitable for a single NVIDIA RTX 4090 (24 GB VRAM).
- Instruct: This is a version fine-tuned by instructions based on the Base model. It is task-oriented and can better complete tasks such as generating and answering questions according to user requests, which reduces the workload required for further fine-tuning and also facilitates our benchmarking.
- Enhanced Code and Math Abilities: According to its developers, the Qwen2.5 series significantly boosts performance on coding and mathematics tasks. These qualities are highly advantageous for our autoformalization objectives.

2.3 Agda

Agda is a dependently typed functional programming language. Due to strong and dependent typing, Agda can also be used as a proof assistant, allowing proofs of mathematical theorems and running proofs such as algorithms. It has many similarities with other functional programming languages and interactive theorem provers based on dependent type theory, such as Coq and Lean.

Agda provides familiar programming constructs, such as inductive data types, pattern matching, let expressions, records, and moduleswith a syntax heavily influenced by Haskell. Unlike systems that separate tactics from terms (e.g. Coq), Agda proofs are written directly in a functional style. Successfully type-checked Agda programs can be extracted to Haskell and compiled to efficient native code via GHC.

Although Agda's expressive power makes it ideal for formal development, its smaller existing codebase (compared to Lean, which is more popular) limits the amount of available human-written formalinformal training data, presenting a huge challenge for machine learning based autoformalization.

2.4 Dedukti

Dedukti is a logic framework based on the modular theory of the -calculus, which can express many theories and logics.

- Rewriting rules: -calculus modulo is a dependently typed -calculus with added type rewriting rules, which can express proofs in modular form. The -calculus is extended with user-defined equation theory, which enables compact representation of different logics.
- Modularity: Logic specifications and proofs can be written in modules, and different theories can be combined through require and namespaces.

In the Informath pipeline, Dedukti formal code is translated into Agda, Coq, and Lean. This unified representation ensures that subsequent processing steps operate on a single, well-typed intermediate language, simplifying the mapping between formal syntax and abstract meaning.

2.5 Grammatical Framework

Grammatical Framework (GF) provides a framework for defining abstract syntax trees (AST) that map to multiple concrete natural languages, thereby supporting translation, parsing, and generation between languages. GF grammars consist of:

- Abstract syntax: Defines a language-independent AST (e.g. mathematical expressions, propositions)
- Concrete syntaxes: Map the AST to multiple natural languages (e.g. English, French, Swedish)

GF's parsing and linearization tools can convert between sentences and ASTs bidirectionally, allowing for controlled generation of paraphrases in any supported language. GF excels in controlled natural language domains, where minimized ambiguity increases both machine interpretability and correctness guarantees.

2.6 Informath

The *Informath* [4] project uses GF and Dedukti to solve the problem of converting mathematical expressions between multiple formal languages and multiple natural languages. The core structure of Informath is a bidirectional pipeline:

- Formal Languages (Agda, Coq and Lean) $\rightarrow$ Dedukti $\rightarrow$ MathCore: Dedukti acts as an interlingua for formal languages. Code written in Agda, Coq, or Lean is first translated into Dedukti. A custom parser then maps Dedukti definitions into the GF MathCore abstract syntax.
- MathCore $\leftrightarrow$ Natural Language: MathCore acts as an interlingua for natural languages. Multiple paraphrases in each target language (English, French, Swedish) are generated by GF, ensuring both grammatical correctness and semantic fidelity.
- Natural Language $\rightarrow$ MathCore $\rightarrow$ Dedukti: The reverse process uses GF to recover the MathCore from a controlled natural sentence, which is then retranslated through Dedukti back to the original formal system.

By design, Informath's pipeline is able to generate a diverse, high-quality, and controlable parallel corpus of formalinformal pairs without relying on expensive human translation or suffering from LLM hallucination. Each generated sentence is guaranteed to correspond precisely to its formal code, making *Informath* an ideal source of supervised data for training LLMs in multilingual autoformalization.

3

Methods

3.1 Datasets and Experimental Groups

The data used in this study are drawn entirely from the self-constructed SMAD dataset. The dataset naming convention follows a structured pattern to describe the type and scope of the data split:

- **SMAD**: Refers to the SMAD dataset.
- **train/test/all**: Indicates whether the set is used for training, testing, or contains all samples without splitting.
- **full**: Contains all formal language-natural language pairs.
- **eng**: Contains only formal language-English pairs.
- **agda**: Contains only Agda-natural language pairs.
- **agda_eng**: Contains only Agda-English pairs.
- **small (optional)**: A smaller subset with 10,000 training samples or 1,000 test samples, used mainly for low-resource or rapid testing scenarios.

For example, `M_SMAD_test_agda_eng_small` indicates a test subset of the SMAD dataset containing only Agda-English pairs on small scale.

We designed six experimental groups (A-F), each trained on different slices of the SMAD dataset for distinct experimental goals:

These groups are designed to test the effects of training across multiple formal languages (A vs. B), multiple natural languages (C vs. D), and varying dataset sizes (A/D vs. F).

3.2 Training Setup

We fine-tuned the pre-trained `Qwen2.5-7B` language model using the `LLaMA-Factory` framework on a single NVIDIA RTX 4090 GPU with 24GB VRAM. LoRA (Low-Rank Adaptation) was applied for fine-tuning, where the original model weights remained frozen and only small, trainable low-rank matrices were added to each layer.

Table 3.1: Experimental Groups Based on SMAD Dataset

Group	Training Data Coverage	Purpose
A	Each FL (Dedukti/Agda/Coq/Lean) ↔ English, trained independently	Baseline: single formal language ↔ English
B	All FL ↔ English, combined training	Effect of joint training of formal languages
C	Agda ↔ Each NL (English/French/Swedish), trained independently	Baseline: single natural language ↔ Agda
D	Agda ↔ All NL combined training	Effect of joint training of natural languages
E	All FL ↔ All NL (12 pairs) combined training	Full multi-pair multilingual model
F	Agda ↔ English in low-resource (10%, 1%) settings	Effect of training data size

The core LoRA parameters used were:

- Rank $r = 8$
- LoRA $\alpha = 32$
- Dropout $= 0.1$

A dynamic learning rate was applied, starting at 5e–5. For all experiments, we used:

- `batch_size` $= 2$
- `gradient_accumulation_steps` $= 8$

This yielded an effective batch size of $2 \times 8 = 16$.

To ensure fairness, the total number of training steps was kept constant across experiments. For instance, the largest data slice (38,736 examples from all FL/All NL) was trained for one epoch (2,421 steps), while the smallest set (3,228 Agda-English examples) was trained for 12 epochs to match the same number of steps (2,421).

3.3 Evaluation Metrics

We evaluate the translation by comparing the formal language code output by the fine-tuned model with the test set text based on SMAD slices. We employed the following standard metrics:

- **BLEU-4**: Measures n-gram precision, emphasizes exact match.
- **ROUGE-1/2/L**: Measures recall-oriented overlap, captures information coverage.
- **Syntax Error Rate**: Percentage of generated formal code that fails to parse as syntactically valid.

To provide a holistic measure, we define a custom composite score:

$$\begin{aligned}\text{SAMD\_Score} = {} & 0.35 \times (1 - \text{ERROR}\%) \times 100 \\ & + 0.35 \times \text{BLEU-4} \\ & + 0.1 \times (\text{ROUGE-1} + \text{ROUGE-2} + \text{ROUGE-L})\end{aligned}$$

This metric balances the paper quality of the translation and the syntax correctness that is critical for practical applications. We use these metrics to compare models and analyze the results of the experiments.

4

Results

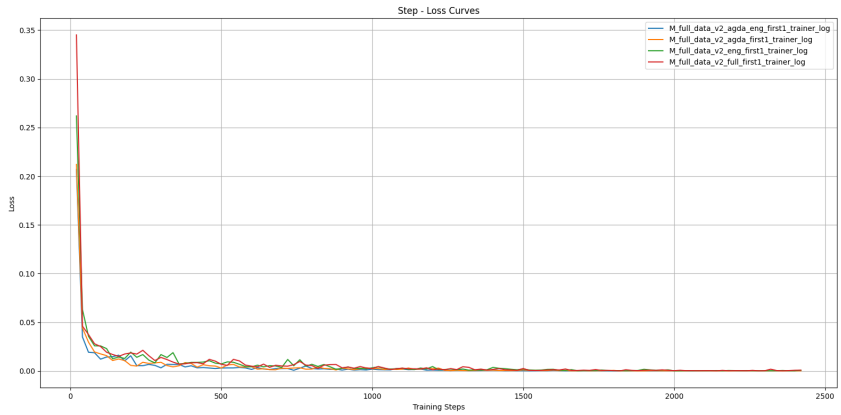

Figure 4.1: Training losses of models using different training sets sliced from SMAD.

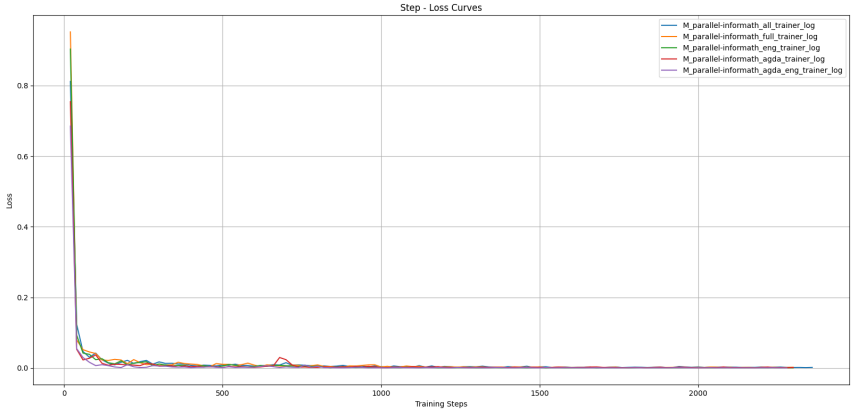

Figure 4.2: Training losses of models using different training sets sliced from Parallel-Informath (data from another resource).

By comparison, we can observe:

- **Baseline vs. fine-tuned models (Nos. 1 vs. 3,6).** The un-fine-tuned baseline achieved a BLEU-4 of 51.20 with a syntax error rate of 83.76%, yielding an overall score of 42.46. In contrast, even the weakest fine-tuned model

Table 4.1: Agda-English test set evaluation (compact)

Model	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L	Syntax Error %	Score
Baseline	51.20	75.08	53.36	60.12	83.76%	42.46
Full data	98.36	99.31	98.94	98.81	3.90%	97.77
Eng first-only	98.63	99.48	99.13	98.99	3.62%	98.01
Agda&Eng first-only	98.91	99.54	99.26	99.27	5.29%	97.58
Parallel full	88.98	92.28	85.85	91.92	7.67%	90.46
Parallel Agda&Eng	85.71	89.62	85.14	89.53	13.04%	86.86

(No. 6) reached BLEU-4 = 85.71 (+67.38%), reduced syntax errors to 13.04%, and achieved overall score = 86.86 (+104.57%). The best fine-tuned variant (No. 3) further lowered syntax errors to 3.62%.

- **Effect of joint training (Nos. 2 vs. 4).** When data volume is large, multi-formal/multi-natural joint training yields similar primary metrics, but it enables the model to translate additional formalisms without degrading Agda-English performance, and incurs negligible extra training cost.
- **Training under low-resource conditions (Nos. 5 vs. 6).** Multi-formal/multi-natural joint training significantly boosts metrics when Agda-English data are scarce, with the most pronounced gain in syntax-error reduction.

5

Conclusion

We achieved excellent BLEU, ROUGE scores and surprising grammatical error rates by fine-tuning Qwen2.5-7B using LoRA on the SMAD dataset. Our work is the first to incorporate Agda into fine-tuning large language models for autoformalization. At the same time, experiments also show that: for the case of scarce Agda resources, multi-language joint training can slightly improve the performance of the model. For the case of abundant data, multi-language joint training greatly improves the efficiency of training without losing training quality. The time originally dedicated to training a single formal language can be used to train 4 different formal languages simultaneously.

A

Appendix 1