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

Autoformalization, translating informal mathematical statements into fully formal machine-checkable code, has recently attracted interest thanks to advances in large language models (LLMs). However, progress has been hindered by the lack of high-quality parallel corpora pairing natural language with formal code, especially for less-popular systems such as Agda. We address this gap by introducing SMAD (Synthetic Multilanguage Autoformalization Dataset), a 400K 4-to-3 parallel corpus covering four formal languages (Dedukti, Agda, Coq, Lean) and three natural languages (English, French, Swedish), generated via the Informath project. We fine-tune the open-source Qwen2.5-7B-Instruct model on SMAD and achieve substantial gains: BLEU-4 improves from 32.90 to 76.16, and Agda syntax error rate falls from 98.43 % to under 8 %. We further explore joint training across multiple formal and natural languages, demonstrating that multilingual and multi-formal regimes yield notable improvements in low-resource Agda settings. Our work establishes the first LLM-based Agda autoformalization system and provides systematic insights into model scaling, multilinguality, and dataset construction for future research.

Keywords: Autoformalization, Multilingual Formalization, Fine-tuning, LoRA, Large Language Models, Qwen2.5-7B-Instruct, Agda, Formal Proof Assistants, Synthetic Multilanguage Autoformalization Dataset (SMAD), GF.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Aarne Ranta, for his invaluable guidance and inspiration throughout this research, and for providing the extensive datasets essential to the project.

My thanks also go to Dr. Josef Urban, who generously spent a great deal of time with us and offered a wealth of expert advice that greatly shaped this work.

I am grateful to Dr. Nick Smallbone for kindly supplying the manually translated "100 Theorems", which proved essential for the experiments and analysis presented here.

Finally, I would like to thank my examiner, Professor Thierry Coquand, for taking the time to review this thesis and for his constructive feedback.

Pei Huang, Gothenburg, 2025-06-05

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Problem Statement 3
1.2 Related Work 3
1.3 Thesis Organization 4

2 Background **5**
2.1 Large Language Models 5
2.1.1 Qwen2.5-7B-Instruct 5
2.1.2 Llama3.1-8B-Instruct 6
2.1.3 Llama3.2-1B-Instruct 6
2.2 Formal Languages 6
2.2.1 Agda 7
2.2.2 Dedukti 7
2.2.3 Lean and Coq 7
2.3 Grammatical Framework 8
2.4 Informath 8

3 Methods **11**
3.1 Datasets 11
3.2 Experimental Groups 12
3.2.1 Model Selection 12
3.2.2 Multi-language Joint Training 12
3.2.3 Ablation Study on Agda Data Ratios 12
3.2.4 Final Model Evaluation 12
3.3 Training and Testing Setups 13
3.4 Evaluation Metrics 13

4 Experiments and Results **15**
4.1 Comparison of Large Language Models 15
4.2 Multi-language Joint Training 17
4.3 Ablation Study on Agda Data Ratios 21
4.4 Final Model 23

4.5 Evaluation on Human-Written Statements 25
4.6 Case Study 26

5 Conclusion **29**
5.1 Summary of Results 29
5.2 Future Work 29

Bibliography **31**

List of Figures

2.1 Overview of the workflow of GF 8
2.2 Overview of the structure of Informah project 9

4.1 Training losses of 3 large language models. 16
4.2 Training losses of models using slices of `joint_training` subset. . . . 19
4.3 Training losses of models using slices of `parallel-informath` subsets. 19
4.4 Training losses of slice1-6 models. 21
4.5 Training losses of fine-tuned models at different epochs. 24

List of Tables

1.1 Examples of SMAD 2

3.1 Experimental Stages and Corresponding SMAD Subsets 12

4.1 Performance of three large language models on different test sets. . . 16
4.2 Models performance comparison of Multi-language Joint Training . . 20
4.3 Agda/Dedukti Code ratios in slices 21
4.4 Models performance of ablation study 22
4.5 Model performance at different training epochs 24
4.6 Comparison of Synthetic vs. Human-Written Statements 25
4.7 Performance on Human-Written Statements 26
4.8 Selected Theorem Outputs and Expert Comments 28

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 named MMA. 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 [2]. 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* [3] 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 400K 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.

Table 1.1 presents a comparison of how the proposition: `For natural numbers, an even number plus one equals an odd number.` is represented across different formal systems (Dedukti, Agda, Coq, Lean) and its corresponding informal mathematical descriptions in English, French, and Swedish.

Table 1.1: Examples of SMAD

Language	Expression(s)
Dedukti	`prop20 : (n : Elem Nat) -> Proof (even n) -> Proof (odd (plus n 1)) .`
Agda	`postulate prop20 : (n : Nat) -> even n -> odd (plus n 1)`
Coq	`Axiom prop20 : forall n : nat, (even n -> odd (n + 1)) .`
Lean	`axiom prop20 (n : Nat) (x : even n) : odd (n + 1)`
English	Prop20. For all instances n of natural numbers, if we can prove that n is even, then we can prove that the sum of n and 1 is odd. Prop20. Let $n \in N$. Assume that n is even. Then $n+1$ is odd.
French	Prop20. Pour toutes les instances n de nombres naturels, si nous pouvons démontrer que n est pair, alors nous pouvons démontrer que la somme de n et de 1 est impaire. Prop20. Soit $n \in N$. Supposons que n est pair. Alors $n+1$ est impair.
Swedish	Prop20. För alla instanser n av naturliga tal, om vi kan bevisa att n är jämnt, så kan vi bevisa att summan av n och 1 är udda. Prop20. Låt $n \in N$. Anta att n är jämnt. Då är $n+1$ udda.

In this work, we fine-tuned the open-source large language model Qwen2.5-7B-Instruct [4] on SMAD and established an autoformalization system for Agda for the first time. To ensure the reliability and generalization of model testing, we use mathematical definitions to construct the training set and more complex mathematical theorems to construct the test set. The BLEU-4 Score of the fine-tuned model on the test set reached 76.16, which was 83.60% higher than the score of 32.90 of the unfine-tuned baseline model. Other metrics including ROUGE-1/2/L also improved by 131.45%/64.35%/240.76% 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 98.43%, 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, the 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.

In general, this work differs from MMA in four key ways:

- Data source: We use the Informath pipeline 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).
- Smaller Models: We fine-tuned a large language model with fewer parameters and achieved better results.

1.1 Problem Statement

This paper aims to build an autoformalization system for Agda by fine-tuning LLMs. At the same time, we also investigate the following research questions:

1. Which model architecture and size strike the best balance between computational cost and autoformalization performance?
2. Does joint training across multiple formal languages improve the quality or efficiency of Agda translation?
3. Does joint training across multiple natural languages provide similar benefits?
4. In low-resource scenarios for Agda, how does the proportion of Agda examples in the training set affect model performance?
5. What types of errors does the fine-tuned model produce in its Agda outputs, and what are the likely underlying causes?

1.2 Related Work

Autoformalization has long been a topic of significant interest. Many early attempts at autoformalization framed the task as neural machine translation. For instance, Wang et al. [5] trained and compared supervised NMT based on RNN, unsupervised NMT, and XLM models on Mizar. More recently, interest in large language models has brought renewed focus to the field: Wu et al. [6] evaluated off-the-shelf LLMs such as PaLM and Codex on a benchmark of mathematical contest problems and found that these models could correctly render 25.3% of examples into Isabelle/HOL without any fine-tuning. However, training or fine-tuning LLMs for autoformalization requires large, high-quality parallel corpora of formal and informal mathematics. To address this, Jiang et al. [1] used GPT-4 to back-translate Isabelle and Lean proofs into English, creating the MMA dataset of 332 K informalformal pairs. While MMAs scale is unprecedented, its noise level remains high: sampling estimates put its translation accuracy at only 74%, and models fine-tuned on MMA produce acceptable proofs just 2931% of the time.

These studies collectively illustrate the promise of neural networks for autoformalization. Our work builds on and extends these directions by:

- Constructing a fully correct, synthetic 4-to-3 parallel corpus (SMAD) via informath.
- Establishing the first LLM-based autoformalization system for Agda.
- Systematically evaluating joint-training across multiple formals and natural languages in low-resource scenarios.

1.3 Thesis Organization

The remainder of this paper is structured as follows:

- **Chapter 2** presents the relevant technical details involved in the paper.
- **Chapter 3** describes dataset construction, experimental design, training and testing procedures, and evaluation metrics.
- **Chapter 4** reports and analyzes the experimental results, including comparisons of different models, multilingual joint training, ablation studies on Agda data proportions, and the autoformalization performance of the final fine-tuned model.
- **Chapter 5** concludes with a summary of experiment findings and an outlook on future work.

2

Background

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 [7] 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.

2.1.1 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.

2.1.2 Llama3.1-8B-Instruct

Llama3.1-8B-Instruct is an 8-billion-parameter instruction-tuned variant of Metas Llama 3 family. It retains the core Transformer design while incorporating improvements in pretraining data diversity and instruction alignment. We include Llama3.1-8B-Instruct to assess how a slightly larger, instruction-tuned model from a different research lineage compares to Qwen2.5-7B-Instruct in autoformalization tasks, especially in terms of translation quality versus computational cost.

2.1.3 Llama3.2-1B-Instruct

Llama3.2-1B-Instruct is a lightweight, 1-billion-parameter instruction-tuned model in the same family, designed for resource-constrained scenarios. By including Llama3.2-1B-Instruct, we evaluate the lower bound of model size necessary for acceptable Agda autoformalization performance and examine trade-offs between model compactness and translation fidelity.

2.2 Formal Languages

Formal languages are precisely defined syntactic and semantic systems used by proof assistants to encode mathematical definitions, propositions, and proofs in a way that machines can check for correctness. By enforcing rigorous typing and inference rules, they eliminate ambiguity inherent in natural language and enable fully verified developments. In this work we consider four such languages: Agda, Dedukti, Lean, and Coq.

2.2.1 Agda

Agda [8] 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.

2.2.2 Dedukti

Dedukti [9] is a logic framework based on the modular theory of the $\lambda\Pi$-calculus, which can express many theories and logics.

- Rewriting rules: $\lambda\Pi$-calculus modulo is a dependently typed λ-calculus with added type rewriting rules, which can express proofs in modular form. The λ-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.

2.2.3 Lean and Coq

Lean [10] and Coq [11] are two of the most widely adopted dependently-typed proof assistants, each supported by a rich ecosystem of libraries, Leans Mathlib and Coqs Standard Library, containing thousands of formalized definitions and theorems. Their popularity and extensive codebases make them natural targets for autoformalization research. By incorporating Lean and Coq alongside Agda in our experiment, we not only advance Agda autoformalization but also enhance the models capacity to generalize across different proof assistants. In future work, the extensive Mathlib and Coqs Standard Library could serve as valuable benchmarks for further evaluation and fine-tuning.

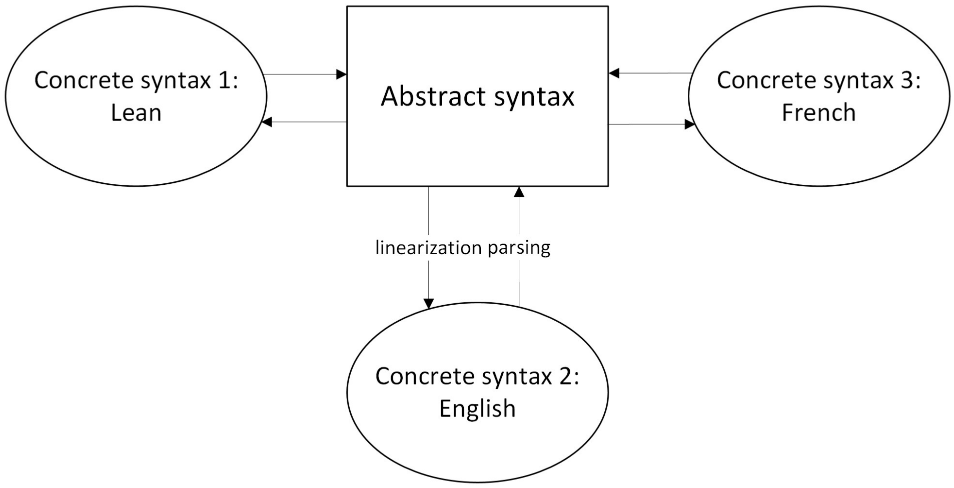

Figure 2.1: Overview of the workflow of GF

2.3 Grammatical Framework

Grammatical Framework (GF) [12] 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.

2.4 Informath

The *Informath* 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) → Dedukti → 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 ↔ 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.

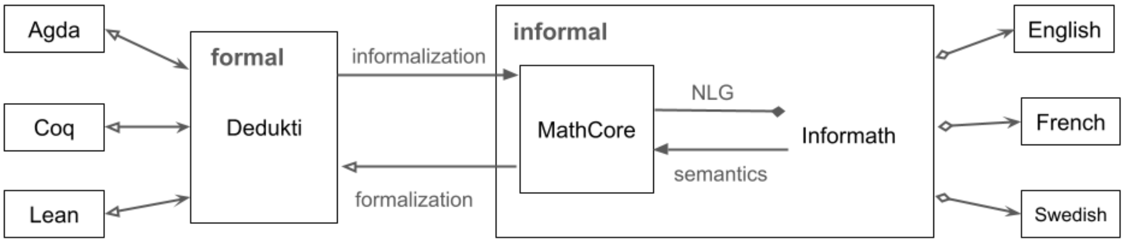

Figure 2.2: Overview of the structure of Informah project

- Natural Language → MathCore → 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.

Here is an example that converts between all currently available languages. The Dedukti statement has been used as the source of all the other formats. Both MathCore and Informath could also be used as source, by parsing them and converting to Dedukti. MathCore renderings are designed to be unique for each Dedukti judgement. But the full Informath language is able to recognize several variations.

```
Dedukti: prop110 : (a : Elem Int) -> (c : Elem Int) ->
  Proof (and (odd a) (odd c)) ->
  Proof (forall Int (b => even (plus (times a b) (times b c)))).

MathCoreEng: Prop110. Let $a$ and $c$ be instances of integers.
↪  Assume that we can prove that $a$ is odd and $c$ is odd. Then we
↪  can prove that for all integers $b$, the sum of the product of
↪  $a$ and $b$ and the product of $b$ and $c$ is even.

InformathEng: Let $a$ and $c$ be integers. Assume that both $a$ and
↪  $c$ are odd. Then for all integers $b$, $a b + b c$ is even.

MathCoreFre: Prop110. Soient $a$ et $c$ des instances d'entiers.
↪  Supposons que nous pouvons démontrer que $a$ est impair et $c$
↪  est impair. Alors nous pouvons démontrer que pour tous les
↪  entiers $b$, la somme du produit de $a$ et de $b$ et du produit
↪  de $b$ et de $c$ est paire.

InformathFre: Prop110. Soient $a$ et $c$ des entiers. Supposons
↪  qu'et $a$ et $c$ sont impairs. Alors pour tous les entiers $b$,
↪  $a b + b c$ est pair.

MathCoreSwe: Prop110. Låt $a$ och $c$ vara instanser av heltal. Anta
↪  att vi kan bevisa att $a$ är udda och $c$ är udda. Då kan vi
↪  bevisa att för alla heltal $b$, är summan av produkten av $a$
↪  och $b$ och produkten av $b$ och $c$ jämn.
```

```
InformathSwe: Prop110. Låt $a$ och $c$ vara heltal. Anta att både
↪  $a$ och $c$ är udda. Då för alla heltal $b$, är $a b + b c$
↪  jämnt.

Agda: postulate prop110 : (a : Int) -> (c : Int) ->
  and (odd a) (odd c) ->
  all Int (\ b -> even (plus (times a b) (times b c)))

Coq: Axiom prop110 : forall a : Int, forall c : Int,
  (odd a /\ odd c -> forall b : Int, even (a * b + b * c)) .

Lean: axiom prop110 (a c : Int) (x : odd a  odd c) :
   b : Int, even (a * b + b * c)
```

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

We name the corpus built by Informath for this project SMAD (Synthetic Multi-language Autoformalization Dataset), which contains four formal languages (Agda, Lean, Coq, Dedukti), each aligned with multiple descriptions in three natural languages (English, French, Swedish). The SMAD corpus is divided into several subsets named model_selection, joint_training, parallel-informath, final_data and natural_statements, which are customized according to different experiments. Unless otherwise stated, each dataset mentioned below refers to one of these predefined SMAD subsets.

The names of the slices for each SMAD subset follow a structured pattern:

- **model_selection**: The name of the subset.
- **train/test**: Indicates the slice is used for training or testing. If the slice name does not have this part, it means that the slice is used for training by default.
- **full**: Contains all formal language-natural language pairs. Default setting.
- **eng**: Contains only formal language-English pairs.
- **agda**: Contains only Agda-natural language pairs.
- **agda_eng**: Contains only Agda-English pairs.
- **first1**: when multiple statements of one natural language exist for one formal language code, keep only the first statement.
- **small**: A smaller subset with 10,000 training samples or 1,000 test samples, used mainly for low-resource or rapid testing scenarios.

For example, `model_selection_test_agda_eng_small` indicates the small-scale test split of the model_selection subset containing only Agda-English pairs. In addition, a leading `M_` signals a model fine-tuned on that split. For example, `M_joint_training_full_first1` is the model trained on the training slice of the joint_training subset containing all formal language-natural language pairs, but for each formal language code, there is only one unique statement in each natural language.

3.2 Experimental Groups

Based on the SMAD corpus introduced in Section 3.1, we organize our experiments into four main stages, each targeting a specific research question. Every experiment draws from one or more predefined SMAD subsets (see Table 3.1). Readers may refer back to Section 3.1 for naming conventions.

Table 3.1: Experimental Stages and Corresponding SMAD Subsets

Stage	SMAD Subset	Size
Model Selection	`model_selection_train_agda_eng_first1`	3154
	`model_selection_test_agda_eng_first1`	351
	`model_selection_test_agda_small`	1000
Joint Training	`joint_training`	390456
	`parallel_informath`	25236
Ratio Ablation	`joint_training` slices with varying Agda ratios	500-2500
Final Evaluation	`final_data`	25860
	`natural_statements_test`	228

3.2.1 Model Selection

Fine-tune each candidate on the same Agda-English single-pair split for three epochs, then evaluate on the two held-out sets to choose the best trade-off between accuracy and resource usage.

3.2.2 Multi-language Joint Training

Two parallel experimentsone mixing all four formal languages with only first statement for each natural language, the other mixing all four formals with all statements for each natural language, to measure the impact of adding linguistic diversity.

3.2.3 Ablation Study on Agda Data Ratios

From the joint_training split, create six low-resource slices with Agda vs. Dedukti in ratios of (500, 0), (500, 500), (125, 500), (500, 2000), (2000, 500), (500, 125). Train each slice once and compare its BLEU-4, ROUGE-1/2/L and syntax error performance.

3.2.4 Final Model Evaluation

Build final_training from only math definitions and final_testing from the "100 Theorem" . Train with multi-formal & multi-natural joint learning for one vs. three epochs, and include a human-translated test set to test model's generalization ability.

Each paragraph above points to a corresponding subsection in Chapter 4, Results. Detailed metrics and analysis are written there.

3.3 Training and Testing Setups

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) [13] was applied for fine-tuning, where the original model weights remained frozen and only small, trainable low-rank matrices were added to each layer.

The core LoRA parameters used were:

- rank $r = 8$
- LoRA $\alpha = 32$
- LoRA$^+$ learning rate $= 16$
- 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 for experiments in the same group was kept constant. For instance, in Multi-language Joint Training group, the largest data slice (38,736 examples from pairs of all formal languages and All natural languages) was trained for one epoch (2,421 steps), while the smaller set (3,228 Agda-English examples) was trained for 12 epochs to match the same number of steps (2,421).

3.4 Evaluation Metrics

To assess the quality of autoformalization, we build on both standard NLP metrics and code-aware measures. In particular, we draw inspiration from *CodeBLEU* [14], which augments BLEU with syntax and data-flow matching to better reflect code correctness and semantic equivalence. While CodeBLEU requires expensive parsing and data-flow analysis, we adopt a lighter composite metric tailored to our setting.

We compare the models generated formal code against the reference SMAD test slices using:

- **BLEU** [15]: n-gram precision with up to 4-grams, emphasizing exact token matches.

- **ROUGE-1/2/L** [16]: Recall-oriented n-gram and longest-common-subsequence overlap, capturing information coverage.
- **Syntax Error Rate**: The fraction of outputs that fail to parse or type-check in Agda, reflecting practical usability.

Inspired by CodeBLEU's multi-facet evaluation, we define *SMAD_Score* as a weighted sum of syntax correctness and surface-level overlap:

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

Here:

- The first term rewards syntactically valid code.
- The second term captures precise token-level matching.
- The third term aggregates information coverage across unigrams, bigrams, and longest common subsequences.

By balancing syntax correctness with both precision and recall aspects of matching, SMAD_Score provides a holistic indicator of translation quality and practical utility. We report all individual metrics alongside SMAD_Score when comparing models and analyzing experimental results.

4

Experiments and Results

4.1 Comparison of Large Language Models

In the initial phase of our experiments, we fine-tuned several open-source large language models to identify which architecture is best suited for Agda autoformalization, while also examining the trade-offs between model size and computational resources (e.g., GPU memory, inference speed). Specifically, we evaluated three instruct-tuned models: Metas Llama-3.1-8B-Instruct, the more recent Llama-3.2-1B-Instruct, and Alibaba's Qwen2.5-7B-Instruct. All training and evaluation data derive from a subset of our SMAD corpus that we designate as `model_selection`.

Each model was fine-tuned on the `model_selection_agda_eng_first1` split for 3 epochs, ensuring equivalent training volume, and subsequently evaluated on two held-out test sets: `model_selection_test_agda_eng_first1` (351 samples) and `model_selection_test_agda_small` (1000 samples). Note that the model_selection subset was created during the data collection and early experimentation phase. These results serve solely as guidance for model selection.

The training losses and evaluation metrics are summarized below. Notice that we apply a moving average with a window size of 5 steps to reduce the high variance inherent in batch-wise loss logging. This smoothing helps highlight general convergence trends while preserving the shape of the learning curve. This processing is applied to all loss figures in this paper.

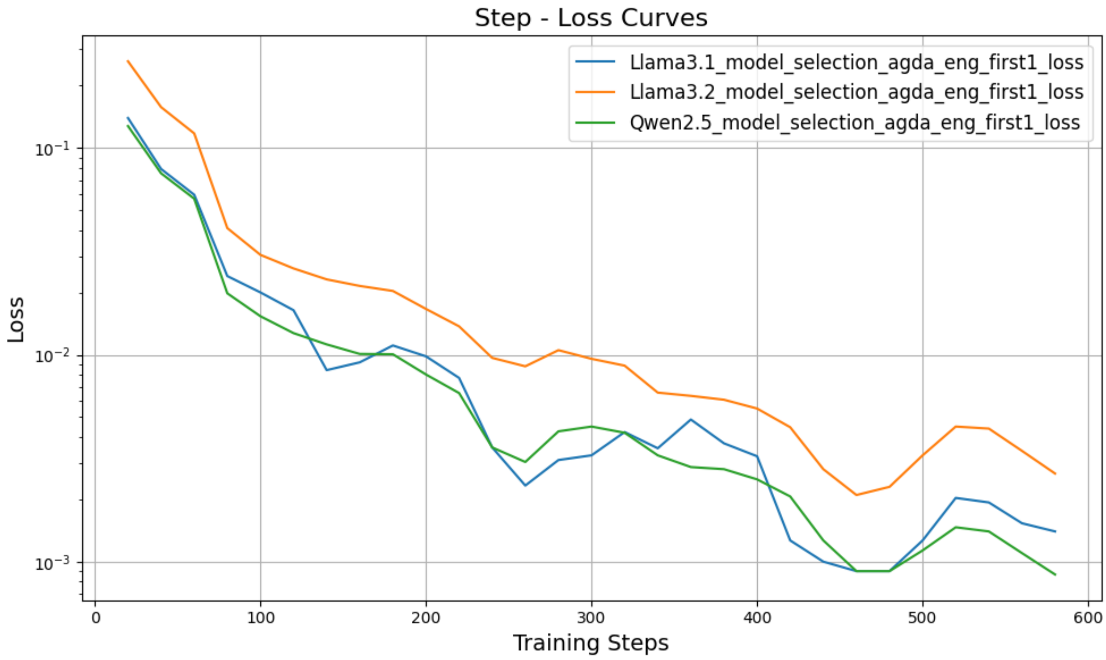

Figure 4.1: Training losses of 3 large language models.

Table 4.1: Performance of three large language models on different test sets.

Model	Testset (Size)	BLEU-4	ROUGE-1/2/L	Syntax Err.%
Qwen2.5	...`first1` (351)	98.62	99.45 / 99.11 / 98.95	2.85
	...`small` (1000)	94.85	97.22 / 95.80 / 96.59	11.10
Llama-3.1	...`first1` (351)	98.67	99.43 / 99.10 / 98.94	3.13
	...`small` (1000)	96.23	97.82 / 96.64 / 97.52	12.20
Llama-3.2	...`first1` (351)	98.29	99.29 / 98.91 / 98.67	5.13
	...`small` (1000)	80.45	89.32 / 82.94 / 85.50	36.10

Considering both performance metrics and model footprint, we selected **Qwen2.5-7B-Instruct** as our primary base model for subsequent experiments, based on the following factors:

1. **High Accuracy with Low Syntax Error Rate**
   - On the smaller test set (`...eng_single`), Qwen2.5-7B-Instruct achieved a BLEU-4 score of 98.62 and ROUGE scores on par with the top performer, while exhibiting the lowest syntax error rate at only 2.85 %.
   - On the larger split (`...small`), it ranked second in overall metricsmarginally behind Llama-3.1-8B-Instructbut maintained a lower syntax error rate (11.10 %) compared to Llama-3.1-8B-Instruct (12.20 %) and dramatically outperformed Llama-3.2-1B-Instruct (36.10 %).
2. **Resource Efficiency**

- With approximately 7.5 billion parameters, Qwen2.5-7B-Instruct demands less GPU memory and achieves faster inference than the 8 billion-parameter Llama-3.1 variant, while delivering substantially better syntax robustness than Llama-3.2-1B-Instruct.
- Models in the 7-8 billion-parameter range demonstrated ample capacity for learning the auto-formalization mappings without incurring prohibitive computational costs.
- Larger models (>8B parameters) introduce steep increases in memory requirements and inference latency, which could hinder experiment reproducibility and practical deployment.

3. **Consideration of More Advance Models**
    - Although newer, larger models (e.g., Llama 4, Qwen 3) have since been released, our dataset contribution and initial experiments concluded prior to their availability. Moreover, existing hardware and time constraints precluded their inclusion in this study. The performance of the current model is already very good, and we can look forward to the results of fine-tuning training of more advanced models.

In summary, Qwen2.5-7B-Instruct represents the optimal balance between translation fidelity, syntactic correctness, and computational efficiency, and thus serves as the foundation for all subsequent auto-formalization experiments.

4.2 Multi-language Joint Training

After selecting Qwen2.5-7B-Instruct as our base model, we conducted further experiments to validate the efficacy of multi-formal-language joint training (as proposed by Jiang et al.) in improving Agda autoformalization, and to further explore the impact of multi-natural-language joint training on the models Agda translation capabilities.

We designed two experimental setups using different SMAD derived sub-datasets, joint_training and parallel-informath (with joint_training encompassing all examples from model_selection). It is also worth noting that, as mentioned in Section 3.2, both experimental datasets in this section are constructed using earlier datasets, and these results only explore the impact of multi-lingual joint training on automatic formalization. Both datasets share identical train/test splitting procedures, but differ in natural-language correspondences:

- joint_training maps each formal code snippet to a single natural-language sentence. Total 3328 Agda-Eng pairs.
- parallel-informath maps each formal snippet to multiple distinct natural-language sentences. Total 1229 Agda-Eng pairs, 152 agda codes are unique.

Their code examples do not overlap, and to equalize total training steps, we adjusted training epochs per data slice.

We also evaluated the latent autoformalization abilities of the un-fine-tuned Qwen2.5-7B-Instruct on both test sets. Without any prompt engineering, the model tends to generate lengthy explanatory text or other irrelevant noise, which severely skews BLEU-4 and ROUGE scoring. Therefore, we refined the original instruction using a one-shot style prompt: we added explicit format and content constraints in the instruction and provided a minimal EnglishAgda example for the model to imitate. An example is shown below:

```
{
    "instruction": "Translate the following English statement into
    ↪  Agda, Please imitate the input and output examples and
    ↪  output in the specified format. Give me the output text only
    ↪  (without any explains, inputs or 'Output:'). Example:
    ↪  Translate the following English statement into Agda. Prop80.
    ↪  We can prove that $2$ is even. Output Should be like:
    ↪  postulate prop80 : even 2  ",
    "input": "Prop30. For all instances $n$ of natural numbers, if
    ↪  we can prove that $n$ is odd, then we can prove that the sum
    ↪  of $n$ and $1$ is even.",
    "output": "postulate prop30 : (n : Nat) -> odd n -> even (plus
    ↪  n 1)"
},
```

Notice that the one-shot example is extremely concise. It simply demonstrates the simple theorem 2 is even. Any large pre-trained model inherently knows this fact. The only purpose of the example is to signal the precise code style and formatting (e.g., use of postulate, arrow notation, spacing). Because it provides no substantive new semantic information, its impact on the un-fine-tuned models autoformalization capability is negligible. Then the output can be used as a baseline to measure the improvement of the subsequent fine-tuned model. All subsequent datasets for testing un-fine-tuned models are designed in this mode to maintain consistency.

The training losses and evaluation metrics are summarized below.

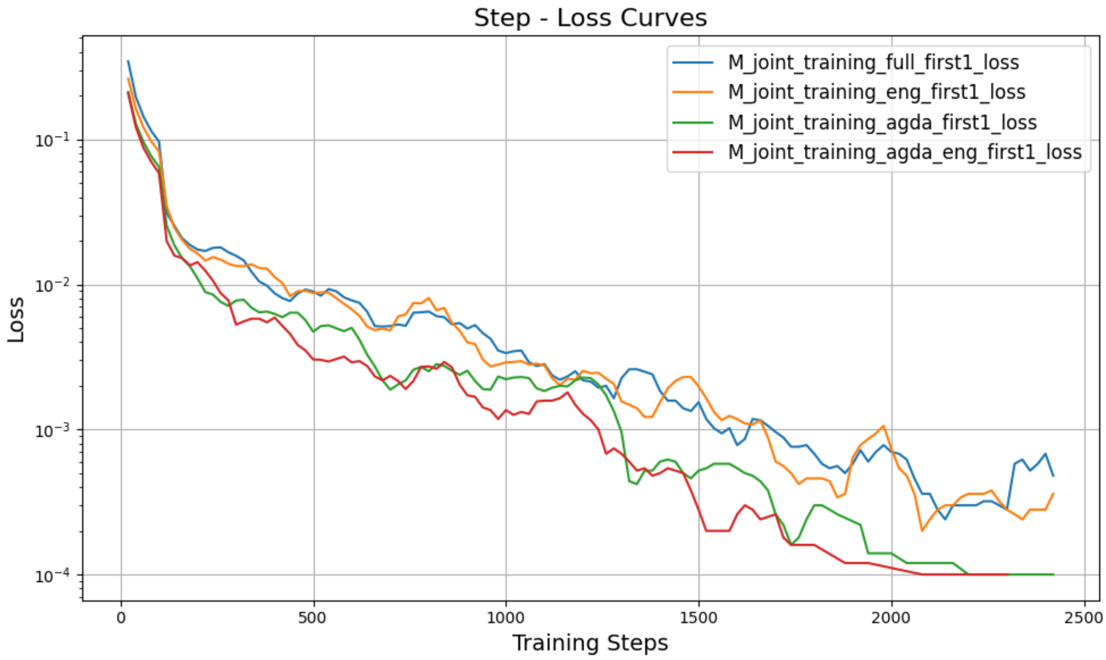

Figure 4.2: Training losses of models using slices of `joint_training` subset.

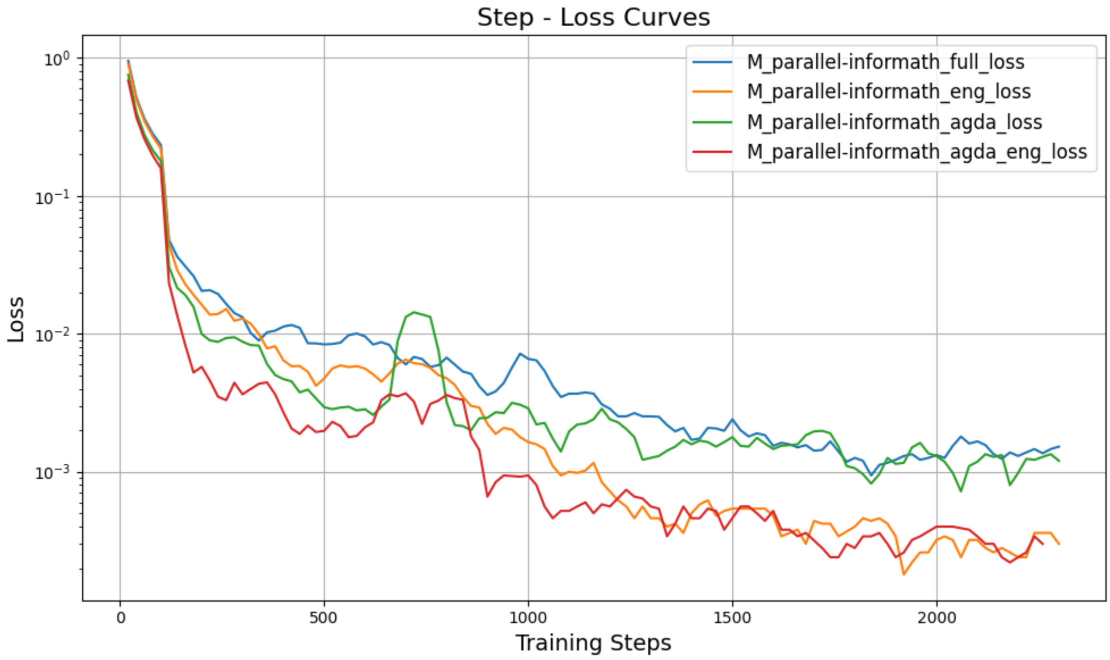

Figure 4.3: Training losses of models using slices of `parallel-informath` subsets.

By comparison, we can observe:

1. **Baseline vs. Fine-tuned Models**
   - The un-fine-tuned baseline achieved a BLEU-4 of 51.20 with a syntax

Table 4.2: Models performance comparison of Multi-language Joint Training

Model	BLEU-4	ROUGE-1/2/L	Syntax Err.%	Score
BaseLine	53.27	71.57 / 50.52 / 60.33	90.25	40.30
M_JT_F1[1]	98.36	99.31 / 98.94 / 98.81	3.90	97.77
M_JT_E1[2]	98.63	99.48 / 99.13 / 98.99	3.62	98.01
M_JT_A1[3]	98.98	99.64 / 99.48 / 99.32	4.74	97.83
M_JT_AE1[4]	98.91	99.54 / 99.26 / 99.27	5.29	97.57
M_PI_F[5]	88.98	92.28 / 85.85 / 91.92	7.67	90.47
M_PI_E	88.61	92.53 / 87.08 / 92.04	9.95	89.70
M_PI_A	88.30	91.04 / 85.33 / 91.06	18.19	86.28
M_PI_AE	85.71	89.62 / 85.14 / 89.53	13.04	86.86

[1]JT_F1: M_joint_training_full_first1
[2]JT_E1: M_joint_training_eng_first1
[3]JT_A1: M_joint_training_agda_first1
[4]JT_AE1: M_joint_training_agda_eng_first1
[5]M_PI_F: M_parallel-informath_full

error rate of 83.76%, yielding a composite score of 42.46, rendering it unusable.

- In contrast, even the weakest fine-tuned model (No. 6) reached BLEU-4 = 85.71 (+67.38%), reduced syntax error rate to 13.04%, and achieved composite score = 86.86 (+104.57%).
- The best fine-tuned variant (No. 3) attains only 3.62% syntax error rate .

2. **Saturation in Large Sample Scenarios**
   - When ample Agda-English data are available, core metrics (BLEU-4 and ROUGE scores) saturate near 98% and syntax-error rates drop below 5%. Although multi-language joint training yields improvements over Agda-English-only training, the gains are limited, indicating that with sufficient Agda-English examples the model already learns an almost perfect mapping.
   - From another perspective, multi-language training does allow the model to generalize to other formal and natural languages without significantly degrading Agda-English performance, and does so at negligible additional training cost.
3. **Complementary Benefits in Low Resources Scenarios**
   - Under data scarcity, multi-formal and multi-natural joint training substantially boosts all performance indicators, most notably reducing syntax-error rate from 13.04% to 7.67%.

This suggests that, when Agda-English pairs are limited, parallel examples in Lean, Coq, Dedukti and in multiple natural languages (e.g., French, Swedish) provide complementary semantic and structural cues that greatly enhance the models autoformalization capabilities.

4.3 Ablation Study on Agda Data Ratios

In the previous section "Joint training of multiple languages", we have observed that parallel examples of multiple formal languages and natural languages can significantly improve performance when Agda-English samples are scarce. In order to deeply analyze the role of Agda data volume in low-resource scenarios, we build multiple slicing experiments with different proportions of formal languages based on the `joint_training dataset` in this section. The number of Agda and Dedukti examples in each training set is as follows:

Table 4.3: Agda/Dedukti Code ratios in slices

Slice	**Agda**	**Dedukti**	**Agda Ratio (%)**
1	500	0	100.00
2	500	500	50.00
3	125	500	20.00
4	500	2000	20.00
5	2000	500	80.00
6	500	125	80.00

The results are as follows:

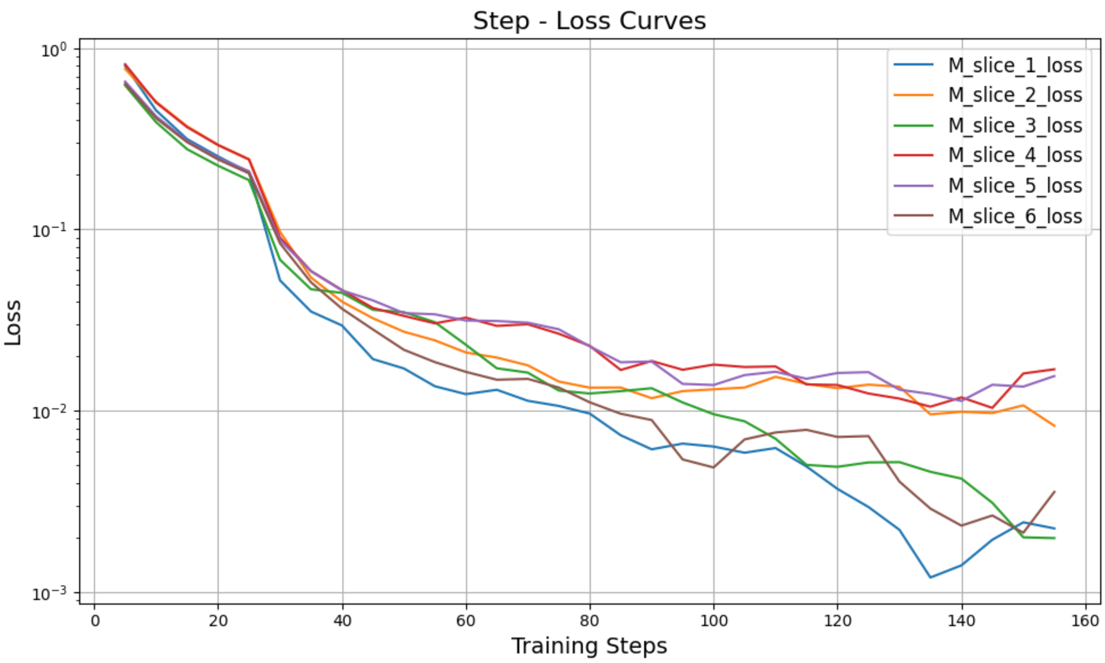

Figure 4.4: Training losses of slice1-6 models.

Table 4.4: Models performance of ablation study

Model	**BLEU-4**	**ROUGE-1/2/L**	**Syntax Err.%**	Score
Baseline	52.56	69.32 / 48.88 / 59.33	93	38.60
M__slice__1	97.73	98.63 / 96.94 / 98.17	5	96.83
M__slice__2	98.35	98.90 / 98.15 / 98.89	4	97.62
M__slice__3	96.74	97.74 / 96.20 / 97.37	10	94.49
M__slice__4	98.11	98.39 / 97.23 / 98.24	7	96.27
M__slice__5	98.59	99.02 / 98.42 / 98.93	5	97.39
M__slice__6	98.18	98.81 / 97.89 / 98.65	2	98.20

We can observe that::

1. Agda-Only (Slice 1): With only 500 Agda examples, the model BLEU-4 jumps to 97.73, and the grammatical error rate drops sharply to 5%. Compared with the baseline (52.56/93%), the single Agda data can greatly improve the performance, indicating that the model has a strong ability to learn Agda autoformalization.

2. Balanced multi-language (slice 2): When the number of Agda and Dedukti examples is equal (500/500), BLEU-4 further improves to 98.35, and the grammatical error rate drops to 4%. The right amount of Dedukti examples can provide additional structural diversity to the model, which helps to improve it slightly.

3. Low Agda and High Dedukti (Slice 3): When Agda is reduced to 125 and Dedukti is kept at 500, the performance is slightly reduced (BLEU-4 96.74, syntax error rate 10%), but it is still far higher than the baseline, indicating that when Agda samples are extremely scarce, Dedukti examples can still greatly help the model effectively learn the autoformalization of Agda.

4. High Dedukti ratio (Slice 4): When Dedukti is greatly increased to 2000 (Agda = 500), the model BLEU-4 is 98.11 and the syntax error rate is 7%. The excessive proportion of Dedukti slightly reduces the grammatical accuracy of Agda, suggesting that non-target language samples cannot be stacked indefinitely.

5. High Agda and Low Dedukti (Slice 5): When Agda is increased to 2000 and Dedukti is 500, the highest BLEU-4 = 98.59, ROUGE-1 = 99.02, and the error rate is 5%. This shows that more Agda samples are still the most direct way to improve core performance.

6. Slight Dedukti Assistance (Slice 6): At a ratio of 500:125, the model's syntax error rate is reduced to the lowest 2%, and BLEU-4 reaches 98.18. This shows that when Agda samples are sufficient, a small amount of Dedukti examples can bring fine-tuning optimization at the format or structure level.

Summary:

- Dominant factor: The number of Agda examples is the primary factor in improving the performance of the core task - from 500 to 2000, BLEU-4 has a steady increase.

- Auxiliary gain: Dedukti data can provide complementary information in scenarios with little or moderate Agda, further reducing syntax errors; but the effect decreases or even regresses when stacked excessively.

4.4 Final Model

In Sections 4.1-4.3 above, each model achieved BLEU-4 and ROUGE scores close to 90-100 on the data generated using Informath; in general natural language translation tasks, BLEU-4 scores of around 70 are considered excellent, and the high scores clearly indicate data bias. As mentioned earlier, these data come from different fields and are divided into training and test sets immediately after being generated by Informath. The generated data may cause the model to learn underlying fixed patterns, resulting in high scores on the test set. Alternatively, because the domain of the data we collected is not wide enough, the model's automatic formalization capabilities are limited to the given domain.

To eliminate this bias, we redesigned the construction logic of the training and test sets:

- For the training set, final_training, it mainly consists of simple mathematical definitions, aiming to allow the model to first master the basic grammatical knowledge of formalized languages, etc.

- The test set final_testing consists of mathematical theorems, which are derived from the "100 Theorem" corpus, evaluating the model's ability to generalize from definitions to proofs.

In addition, in order to further test the generalization ability of the model, we also manually compiled a control dataset. Its formal code is the same as final_testing, but the natural language is manually translated.

Based on the experimental results of 4.2 and 4.3, we adopt multi-formal language and multi-natural language joint training in final_training, and the proportion of each formal language in the dataset is consistent, so as to maximize the automatic formalization ability of our model.

The training results are as follows:

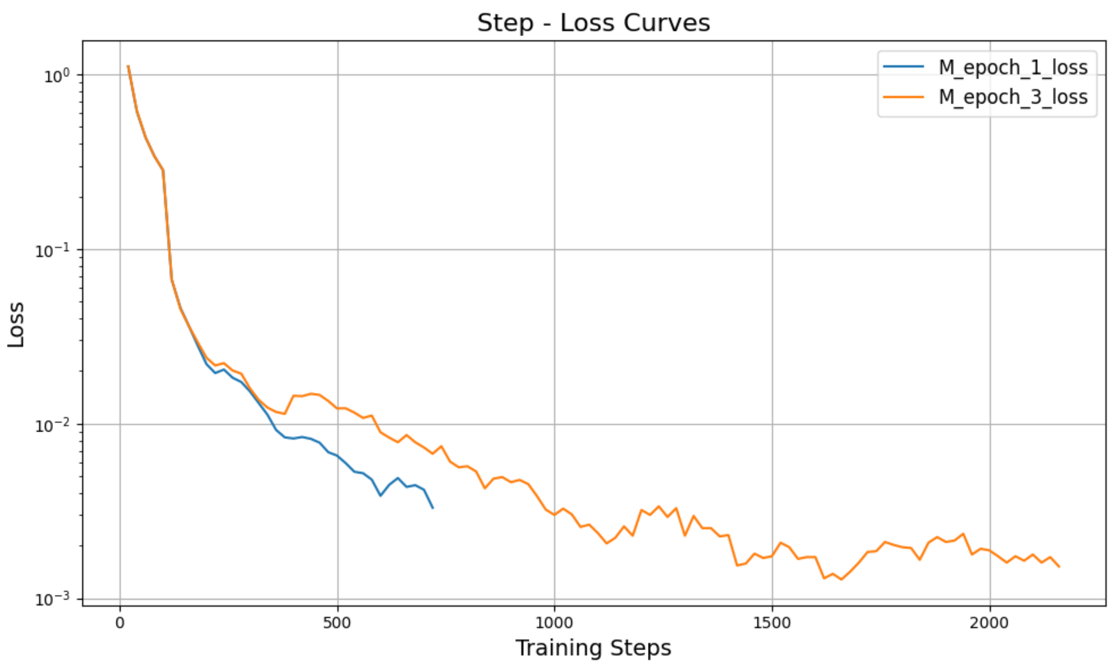

Figure 4.5: Training losses of fine-tuned models at different epochs.

Table 4.5: Model performance at different training epochs

Model	BLEU-4	ROUGE-1/2/L	Syntax Err.%	Score
Baseline	32.90	54.17 / 21.99 / 42.76	98.43	23.96
M_epoch_1	76.16	89.03 / 74.94 / 83.22	7.93	83.60
M_epoch_3	77.78	89.86 / 76.63 / 84.37	20.48	80.14

1. Single epoch training has the best effect
    - Although the BLEU-4/ROUGE scores obtained by the model that only trained one epoch are lower than those of the previous models, the most core indicator syntax-error rate has not decreased much compared with M_parallel-informath_full.
    - The drop in BLEU-4/ROUGE scores is acceptable, because the performance of the baseline model on final_testing has also dropped significantly compared to its performance in section 4.2. Comparing BLEU-4/ROUGE Δ%, it can be found that the model has a greater improvement than all previous models trained based on joint_training. This shows that this dataset reconstruction effectively stimulates the model's ability to transfer learning from definitions to theorems.
2. Over-training leads to overfitting or semantic shift
    - Unfortunately, although the model trained for 3 epochs has a slight improvement in BLEU-4/ROUGE scores, its syntax-error rate has dropped severely, forcing 20.5%.

- We believe that this is likely to be due to the model overfitting in more training steps, or learning more non-Agda expression patterns, which has damaged the performance of the core task.

The reconstructed dataset significantly alleviates the bias problem of the previous dataset and enables the model to obtain a more realistic ability evaluation in more challenging scenarios. One epoch of joint training can maximize the learning gain, while further training may hurt performance due to overfitting or interference from non-target languages.

4.5 Evaluation on Human-Written Statements

At the end of our experiments, we evaluated several representative models on human-written natural-language statements to test their true generalization. The table below contrasts synthetic Informath statements with relatively concise manual sentences:

Table 4.6: Comparison of Synthetic vs. Human-Written Statements

Agda Code	**Informath**	**Human-Written**
`postulate Thm01 : (m : Nat) (n : Nat) Neq n 0 Neq (pow (div m n) 2) 2`	Thm01. Let m and n be instances of natural numbers. Assume that we can prove that n is not equal to 0. Then we can prove that the exponentiation of the quotient of m and n and 2 is not equal to 2. Thm01. Let $m, n \in N$. Assume that $n \neq 0$. Then $(\frac{m}{n})^2 \neq 2$.	Thm01. If $m \in \mathbb{N}$ and $n \in \mathbb{N}^+$ then $(m/n)^2 \neq 2$.
`postulate thm09 : (c : Circle) (r : Real) Eq r (radius c) Eq (area c) (times pi (pow r 2))`	Thm09. Let c be a circle. Let r be an instance of real numbers. Assume that we can prove that r is equal to the radius of c. Then we can prove that the area of c is equal to the product of the number π and the exponentiation of r and 2. Thm09. Let c be a circle. Let $r \in R$. Assume that r is equal to the radius of c. Then the area of c is equal to πr^2.	Thm09. Any circle of radius r has area πr^2.

We can observe that:

- The BLEU-4/ROUGE scores of the model previously trained on joint_training have dropped significantly, with BLEU-4 dropping from nearly 98% to 60-65. The Syntax Error rate has also dropped from 3-5% to 17%. Although the

Table 4.7: Performance on Human-Written Statements

Model	BLEU-4	ROUGE-/2/L	Syntax Err.%	Score
Baseline	32.84	44.83 / 15.86 / 37.85	92.98	23.80
M_JT_F1	60.37	74.00 / 52.22 / 68.12	17.54	69.42
M_JT_AE1	65.00	76.66 / 55.48 / 70.90	17.54	71.91
M_epoch_1	68.67	79.61 / 61.97 / 73.84	8.77	77.50
M_epoch_3	65.56	78.03 / 59.66 / 70.91	22.81	70.83

performance has dropped so severely, it is still significantly improved compared to the un-fine-tuned model.

- The final model shows strong generalization ability, the BLEU-4/ROUGE score does not drop significantly, and the syntax error rate is almost the same as the performance on the original test set
- It is worth noting that the performance gap between M_epoch_1 and M_epoch_3 has significantly widened, and M_epoch _3's previous insignificant leading advantage in BLEU-4/ROUGE scores has disappeared, fully demonstrating that its generalization ability is not as good as M_epoch_1, indicating that our previous judgment that it is overfitting or that the performance is backfired due to interference from non-target languages is accurate.

4.6 Case Study

To better understand the remaining failure modes of our fine-tuned model, we examined several representative theorems and compared the output of M_epoch_3 on human-written statements against the label. Table 4.8 lists selected examples along with their BLEU-4 scores and expert comments. From these examples and the baseline behavior on the same test set, we identify three main error categories:

Error Categories and Causes

1. **Unsupported Infix Operators:** The model frequently emits operators like `==`, `<=`, `>` that are not defined in Agda core library. These originate from the pre-training corpus (Maybe from Lean) and, though familiar to the model, cause type errors in vanilla Agda.
2. **Hallucinated Identifiers:** Terms like `powerset2` do not exist in either the library or the test set. Such "invented" names suggest the model has learned spurious patterns that merge or abbreviate known constructs but do not correspond to valid definitions.
3. **Predicate Misuse:** In several theorems (e.g. Thm78a, Thm22 second variant), the model substitutes generic numeric operations (`plus`) for domain-specific functions (`vectorPlus`), or misuses a morally correct but not in scope

`uncountable`. This reflects a mismatch between the pre-learned "general expression" and the projects formalization conventions.

While many model outputs exhibit clear errors, some (such as the first variant of Thm22) convey the correct mathematical idea in a type-safe manner, even though they diverge from the target theorems logical form. These predictions would compile successfully but score poorly on automatic metrics like BLEU due to lexical and structural differences.

To further assess the practical usability of model outputs involving unsupported infix operators, we normalized these operators in the fine-tuned model's predictions into their equivalent prefix forms or function calls compatible with standard Agda syntax. After this simple normalization step, the number of syntax errors dropped significantly from 13 to just 2. Moreover, 23 out of 57 predictions became type-correct, representing a substantial improvement over the original results. This suggests that many of the model outputs are structurally close to being correct and can be made fully functional with minimal post-processing. This result suggests that many model outputs are easily correctable, making them promising candidates for integration into practical formalization systems.

Table 4.8: Selected Theorem Outputs and Expert Comments

Thm.	Model Output	Label	BLEU	Expert Comment
20a	`postulate thm20a : (p : Nat) -> prime p -> congruent p 1 4 -> exists Nat ( x -> exists Nat ( y -> and (square x) (square y) (equal (plus (times x x) (times y y)) p)))`	`postulate thm20a : (p : Nat) -> prime p -> (k : Nat) -> Eq p (plus (times 4 k) 1) -> exists Nat ( x -> exists Nat ( y -> Eq p (plus (square x) (square y)))`	63.97	*Expert: Clever use of* ***`congruent`****, but the final part will not type-check.*
22	`postulate thm22 : exists Real ( x -> not (exists Real ( y -> and (Lt y x) (Lt x y))))`	`postulate thm22 : not (denumerable Real)`	5.72	*Expert: Semantically different, though type-correct.*
22	`postulate thm22 : uncountable Real`	`postulate thm22 : not (denumerable Real)`	23.45	*Expert: Uncountable is meaningful but not in scope.*
52	`postulate thm52 : (A : Set) -> finite A -> equalset (powerset A) (powerset2 A)`	`postulate thm52 : (A : Set) -> finite A -> Eq (cardinality (powerset A)) (pow 2 (cardinality A))`	54.53	*Expert: "powerset2" is a typical hallucination;* ***`equalset`*** *is wrong predicate for equality of cardinality.*
78a	`postulate thm78a : (u : Vector) -> (v : Vector) -> orthogonal u v -> dotProduct u v == 0`	`postulate thm78a : (u : Vector) -> (v : Vector) -> if (orthogonal u v) (Eq (dotProduct u v) 0)`	60.19	*Expert: Model uses a function type, where Label uses the if operator. Equivalent logic but uses unsupported* `==` *in Agda library.*
91	`postulate thm91 : (u : Vector) -> (v : Vector) -> length (plus u v) <= plus (length u) (length v)`	`postulate thm91 : (u : Vector) -> (v : Vector) -> Leq (norm (vectorPlus u v)) (plus (norm u) (norm v))`	55.77	*Expert: Correct meaning, but misuses* ***`plus`*** *(defined for numbers but not for vectors) instead of vectorPlus and uses undefined operator* `<=`*.*

5

Conclusion

5.1 Summary of Results

We have presented SMAD, a 400K -pair multilingual autoformalization corpus covering four proof assistants (Dedukti, Agda, Coq, Lean) and three natural languages (English, French, Swedish), all derived via the Informath pipeline to guarantee logical correctness and paraphrase diversity.

We achieved excellent BLEU, ROUGE scores and surprising grammatical error rates by fine-tuning Qwen2.5-7B-Instruct 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.

5.2 Future Work

Looking ahead, we plan to extend and deepen this research in several directions:

- **Larger and Newer Models:** Fine-tune with more powerful LLMs (e.g. Llama 4, Qwen 3 series).
- **Human-Translated Benchmarks:** Test on a larger, manually curated formalinformal corpus to assess true generalization beyond synthetic data.
- **Extended Formal and Natural Coverage:** Incorporate additional proof assistants (e.g. HOL, Isabelle) and natural languages (e.g. German, Spanish, Chinese) to broaden applicability and further test multilingual joint training.
- **Bidirectional Translation:** Investigate informalization capabilities of the fine-tuned models.

Bibliography

[1] A. Q. Jiang, W. Li, and M. Jamnik, "Multi-language diversity benefits autoformalization," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, *et al.*, Eds., vol. 37, Curran Associates, Inc., 2024, pp. 83 600–83 626. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2024/file/984de836e696ba653bbfbbbfce31d3bc-Paper-Conference.pdf`.

[2] G. Perkovi, A. Drobnjak, and I. Botiki, "Hallucinations in llms: Understanding and addressing challenges," in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, IEEE, 2024, pp. 2084–2088.

[3] A. Ranta. "Informath." GitHub repository. (2025), [Online]. Available: `https://github.com/aarneranta/informath/tree/main` (visited on 05/29/2025).

[4] Qwen, : A. Yang, *et al.*, *Qwen2.5 technical report*, 2025. arXiv: `2412.15115 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2412.15115`.

[5] Q. Wang, C. Brown, C. Kaliszyk, and J. Urban, "Exploration of neural machine translation in autoformalization of mathematics in mizar," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2020, New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 85–98, ISBN: 9781450370974. DOI: `10.1145/3372885.3373827`. [Online]. Available: `https://doi.org/10.1145/3372885.3373827`.

[6] Y. Wu, A. Q. Jiang, W. Li, *et al.*, "Autoformalization with large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 353–32 368, 2022.

[7] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[8] A. Bove, P. Dybjer, and U. Norell, "A brief overview of agda–a functional language with dependent types," in *Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings 22*, Springer, 2009, pp. 73–78.

[9] A. Assaf, G. Burel, R. Cauderlier, *et al.*, "Dedukti: A logical framework based on the $\lambda\Pi$-calculus modulo theory," 2016.

[10] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, "The lean theorem prover (system description)," in *Automated Deduction - CADE-25*,

A. P. Felty and A. Middeldorp, Eds., Cham: Springer International Publishing, 2015, pp. 378–388, ISBN: 978-3-319-21401-6.

[11] P. Letouzey, "Extraction in coq: An overview," in *Logic and Theory of Algorithms: 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008 Proceedings 4*, Springer, 2008, pp. 359–369.

[12] A. Ranta, "Grammatical framework," *Journal of Functional Programming*, vol. 14, no. 2, pp. 145–189, 2004.

[13] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.

[14] S. Ren, D. Guo, S. Lu, *et al.*, "Codebleu: A method for automatic evaluation of code synthesis," *arXiv preprint arXiv:2009.10297*, 2020.

[15] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds., Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. DOI: `10.3115/1073083.1073135`. [Online]. Available: `https://aclanthology.org/P02-1040/`.

[16] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: `https://aclanthology.org/W04-1013/`.