Multilingual Autoformalization via Fine-tuning Large Language Models with Symbolically Generated Data

Pei Huang, Nicholas Smallbone, Aarne Ranta

December 12, 2025

https://scml.risc.jku.at

RISC Proceedings on
Symbolic Computation and Machine Learning
Publication Number: 1

doi.org/10.35011/risc-proceedings-scml.1

Cite as: Pei Huang, Nicholas Smallbone, Aarne Ranta: Multilingual Autoformalization via Fine-tuning Large Language Models with Symbolically Generated Data. *RISC Proceedings on Symbolic Computation and Machine Learning*, 1, December 12, 2025. SCML, https://scml.risc.jku.at, doi.org/10.35011/risc-proceedings-scml.1.

The SCML publication forum is dedicated to all research that strives to combine "Symbolic Computation" (SC) and "Machine Learning" (ML) as two major approaches to "Artificial Intelligence", in particular to the application of ML to SC, the application of SC to ML, and the hybrid combination of SC and ML to solving problems. More information about SCML is available at https://scml.risc.jku.at.

Steering Commitee and Editorial Board

Scientific Commitee

Multilingual Autoformalization via Fine-Tuning Large Language Models with Symbolically Generated Data

Pei Huang, Nicholas Smallbone, and Aarne Ranta

Department of Computer Science and Engineering,
Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden
{peih,nicsma,aarne}@chalmers.se

Abstract

Autoformalization, translating informal mathematical text into formal machine-checkable code, has recently attracted new interest thanks to advances in large language models. However, progress has been hindered by the lack of high-quality parallel corpora pairing natural language with formal code, especially for less widely used systems such as Agda. In this work, we address this gap by demonstrating a systematic data generation pipeline via the Informath project based on Grammatical Framework on the informal side and Dedukti on the formal side. Using this method, we construct a 400K "4-to-3" parallel corpus spanning four formal languages (Dedukti, Agda, Rocq, Lean) and three natural languages (English, French, Swedish). To validate the utility of the approach, we fine-tune the open-source Qwen2.5-7B-Instruct model 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 scenarios. Our work establishes an easily extensible multilingual autoformalization system and a test dataset with hand-written theorem statements paired with Agda, Dedukti, Lean, and Rocq. We will also propose some systematic insights into dataset construction and multilingual training for future research.

1 Introduction

Autoformalization is the task of translating mathematical natural language into fully formalized, machine-verifiable code. Many recent attempts at autoformalization are based on training or fine-tuning large language models (LLMs). One of the major challenges in these approaches is the lack of large-scale, high-quality parallel corpora that align formal language codes with their informal mathematical statements. This difficulty is faced by all formal languages, including popular ones such as Lean [8] and Rocq (formerly Coq) [4], but in particular less used ones such as Agda [5].

Recent efforts have attempted to address this issue by leveraging LLMs themselves to construct synthetic datasets. Notably, Jiang et al. [13] introduced the MMA dataset, consisting of 332,000 pairs of formal and informal mathematical statements created by using GPT-4 to translate existing formal code in Lean and Isabelle into English. While this approach is cost-effective and scalable compared to manual annotation by well-trained experts, the generated informal text contains considerable noise and incorrect translations. Furthermore, the risk of hallucinations in the data output by large language models cannot be ignored [17].

In contrast to the noisy synthetic pipelines using LLMs, we propose a symbolic approach using the *Informath* [22] project based on Grammatical Framework (GF) [19, 23] and Dedukti [1], which builds multilingual informal-formal pairs from verified formal sources. GF is an interlingual grammar formalism, where a shared abstract syntax can be mapped to multiple natural languages and back. Dedukti is, likewise, a type-theoretical core language to which

several other formalisms (including Agda, Lean, and Rocq) can be translated. The Informath project adds to this back-translations from Dedukti to Agda, Lean and Rocq, which are partial (because Dedukti is more liberal than the other formalisms), but sufficient for the kind of data this project deals with so that correctness in the target formalisms can be guaranteed. By combining Dedukti with GF, we can use any set of formal data in any of the supported formalisms to symbolically generate a parallel corpus of formal-informal pairs.

In the experiments reported here, we have built two such corpora. In both corpora, each Dedukti expression is translated into three additional formal systems (Agda, Rocq, and Lean) and multiple paraphrases in three natural languages: English, French, and Swedish. This design yields a 4-to-3 formal-informal mapping with high correctness and alignment across languages. We have used the corpora to train LLM-based models for autoformalization, and tested the models on symbolically generated theorems, as well as 57 hand-written English statements for a subset of the "100 theorems" of [28].

The first corpus we have built is SMAD (Synthetic Multilanguage Autoformalization Dataset), which includes over 400,000 formally verified examples from existing libraries of different formal systems, automatically converted to Dedukti. For building the autoformalization model, we divided SMAD into training, validation, and test sets in the usual way (8:1:1). Table 1 shows an example from SMAD of a simple statement in Dedukti, Agda, Rocq, Lean on the formal side and in English, French, and Swedish on the natural language side. Two informal statements are shown for each language: a baseline informalization that is in one-to-one correspondence with the formal, and an alternative that is closer to natural mathematical text. All statements (also the formal ones) are automatically generated from the Dedukti source by compiler-like symbolic functions. On the informal side, the full dataset contains up to hundreds of alternative verbalizations per formal statement.

When training autoformalization models on SMAD, we observed that it is difficult to prevent the model from overfitting. To mitigate this, we designed a second corpus with a very different training methodology. This corpus is smaller, consisting of just 198 Dedukti definitions and theorem statements, converted to 8620 variant informal statements, resulting in 25860 formal-informal pairs. Rather than a traditional randomized train-test split, we divide the corpus into *definitions* and *theorem statements*. Definitions are used as training data, and theorem statements are used as test data. This set-up is not unlike the conditions of human mathematicians: learning the definitions of concepts is sufficient for understanding their use in theorem statements. We found that a model trained using this method generalizes better and that, surprisingly, the small number of formal statements suffices to train the model.

While Lean and Rocq have received growing attention in the machine learning community, Agda has remained underexplored due to the scarcity of available training data. Our work is the first to construct an LLM-based autoformalization system that addresses Agda. Evaluating with Agda is a way to see how well our methodology works in a low-resource setting.

As a more general aim, we explore whether joint training across multiple formal and/or natural languages can improve the performance and training efficiency of Agda autoformalization. Specifically, we examine: (1) how incorporating multiple formal systems (Lean, Rocq, and Dedukti) influences the quality of Agda translations, (2) whether multilingual supervision (in English, French, and Swedish) enhances the ability and generalization of the model, and (3) how the proportion of Agda examples in the training set affects the model performance in Agda code low-resource scenarios. Our work differs from existing studies in four major aspects:

1. Data source: We generate formal $\leftrightarrow$ natural pairs with controlled paraphrase diversity using the *Informath* pipeline. This is in contrast both with earlier symbolic approaches, which use limited controlled languages, and with noisy LLM-generated back-translation.

Table 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)`
Rocq	`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.

What is more, the pipeline enables the symbolic generation of new formal-informal data whenever formal data is available.

2. Ours is the first work to include Agda and Dedukti.

3. Multilingual coverage: Our dataset includes three natural languages (English, French, Swedish), enabling experiments with multiple languages on the informal side.

4. Lightweight models: We achieve good results even by fine-tuning a relatively small open-source model, improving accessibility and reproducibility.

5. Small compute footprint: The SMAD corpus can be produced in less than 30 seconds of CPU time on a Macbook Air M2.

By demonstrating successful fine-tuning on Agda, this work lays the foundation for future research on the autoformalization between formal languages and multilingual formalisms in low-resource scenarios.

2 Related Work

The early history of autoformalization features symbolic systems such as Mizar [3], GF-Alfa [11], ForTheL [16], and Naproche [7]. The first neural attempts framed the task as neural machine translation (NMT). Thus Wang et al. [27] trained and compared supervised NMT based on RNN, unsupervised NMT, and XLM models on Mizar. More recently, interest in LLMs has brought renewed focus to the field: Wu et al. [29] 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

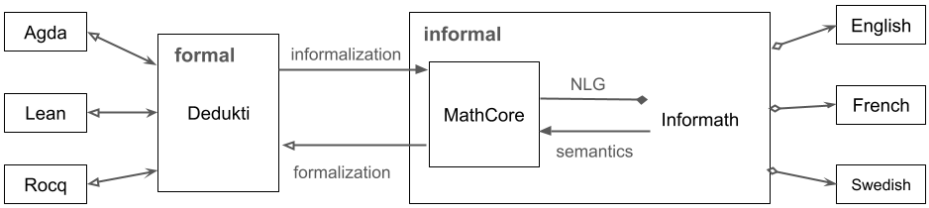


Figure 1: Overview of the structure of Informath. Hollow arrowheads mark partial functions, and diamond-shaped arrowheads mark operations that can give more than one result. Thus the hollow diamond-shaped conversions from actual natural languages to Informath indicate that the coverage of the grammar is bound to be incomplete.

of formal and informal mathematics. To address this, Jiang et al. [13] used GPT-4 to back-translate Isabelle and Lean proofs into English, creating the MMA dataset of 332 K informal-formal pairs. While MMA's scale is unprecedented, its noise level remains high: sampling estimates put its translation accuracy at only approximately 74%, and models fine-tuned on MMA produce just 29–31% acceptable code.

3 Methods

3.1 Informath

The dataset of this project is generated in the *Informath* [22] project, which uses GF and Dedukti to solve the problem of converting mathematical expressions between multiple formal languages and multiple natural languages. Informath's core structure is a bidirectional pipeline:

- Formal Languages (Agda, Rocq and Lean) → Dedukti → MathCore: Dedukti acts as an interlingua for formal languages. Code written in Agda, Rocq, or Lean can be translated into Dedukti. A compiler-like code generator maps Dedukti code into an abstract syntax of natural language in GF.
- Abstract Syntax ↔ Natural Language: Multiple paraphrases are generated in abstract syntax and linearized to different target languages (English, French, Swedish) by GF, ensuring both grammatical correctness and semantic fidelity.
- Natural Language → Dedukti: The reverse process uses GF parsing to convert texts to abstract syntax trees, which are translated through Dedukti to other formal systems.

By design, Informath's pipeline enables the generation of a diverse, high-quality, and controllable parallel corpus of formal-informal pairs without suffering from LLM hallucination. Each generated sentence is guaranteed to correspond to its associated formal code. Readers can refer to Table 1 for an example. However, the opposite direction of autoformalization in pure Informath can only be partial, because no grammar can be guaranteed to cover all natural language. By fine-tuning large language models on data generated by Informath, we can extend the grammar-based parser to a system that suggests a formalization for any natural language input. The resulting formalizations are less certain than grammar-based ones, but their validity can be checked by humans via symbolic feed-back informalizations *without the need to look at the formal code*.

The Informath pipeline follows the methodology described in [20], which in turn was inspired by classical techniques of natural language generation (NLG) [24]. The starting point is MathCore, a controlled natural language (CNL), which is a direct, unambiguous verbalization of Dedukti. Inspired by [10], Informath aims to extend MathCore into a more comprehensive grammar of the language of mathematics. This variation is created effortlessly by using the GF Resource Grammar Library, which is a wide-coverage grammar taking care of the details of morphology and syntactic structures [19]. The different steps of the pipeline can be shown in a verbose mode to trace the exact correspondences between the formal and informal languages. The main intermediate representations are: abstract syntax tree of Dedukti code, abstract syntax tree of GF core language, abstract syntax trees of NLG-generated paraphrases, tokenized natural language output.

Informath can informalize any Dedukti code and thereby any code in other systems (including Agda, Lean, and Rocq) that have conversions to Dedukti. However, the quality of informalization depends on the availability of a lexicon, that is, mappings from formal identifiers to natural language expressions. Informath has methods for automating much of this by using sources such as Wikidata [26] when the concepts belong to standard mathematics [21]. For more advanced research mathematics, either a human has to provide a lexicon or simple heuristics are used, which means that the readability can suffer.

The back-translations from Dedukti to other formal systems are bound to be partial, because Dedukti is more permissive than any of them. This concerns in particular proofs, where the translation of a Dedukti proof may fail to type-check in the other systems. Moreover, the back-translations are not guaranteed to use the target formalisms in the most idiomatic ways, as can already be seen in the examples we show. The goal has just been to generate code that is type-correct, which is enough in scenarios where proof systems are used as checkers of informal mathematics and human readers are not expected to read the formal code.

The data studied in this paper does not include proofs, but only definitions and theorem statements, mostly restricted to standard undergraduate-level mathematics, from [6, 28, 2] (formalized manually, then informalized automatically) and from the Dedukti version of Matita arithmetic library ([1] Section 8.4, informalized automatically). This restriction is similar to the MMA data in [13], but the set of concepts covered in our data is even more limited. Research-level mathematics is still rarely addressed in autoformalization literature [18].

3.2 Experiment Design

The SMAD corpus consists of two datasets named `joint_training` and `parallel_informath` (the datasets are described in Section 4). The second, smaller corpus introduced earlier is used for training and evaluating the final models, and includes `final_data` and `natural_statements`. Based on this corpus, we organize our experiments into three main stages, each targeting a specific research question. Every experiment draws from one or more datasets (see Table 2).

In the **joint training** stage, we compare two variants mixing four formal languages: one with a single natural-language statement per code, and another using all available statements, to examine the impact of linguistic diversity. In the **ratio ablation** stage, we construct six slices from `joint_training` with different Agda-to-Dedukti ratios, and evaluate performance using a composite of BLEU [15], ROUGE [14], and syntactic correctness. The **final evaluation** stage uses a training set of mathematical definitions, and a test set based on the "100 theorems" [28], to assess generalization. Each stage will be described in a corresponding subsection of Section 4, where metrics and analyses are presented in detail.

Table 2: Experimental Stages and Corresponding Datasets

Stage	Dataset(s)	Size
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`	228

The datasets used in the experiments are available via the Informath Git repository[1]. Other datasets will be made available in the same place.

3.3 Training and Testing Setups

All experiments were conducted by fine-tuning the pre-trained `Qwen2.5-7B-Instruct` model using the `LLaMA-Factory` framework on a single NVIDIA RTX 4090 GPU (24GB VRAM). We applied LoRA (Low-Rank Adaptation) [12] by freezing the baseline model and inserting trainable rank-8 matrices ($r = 8$) into each layer.

Before large-scale training, we conducted small pilot experiments and employed grid search combined with empirical validation to identify the most suitable hyperparameter configuration. The final settings included LoRA $\alpha = 32$, LoRA$^+$ learning rate $= 16$, dropout $= 0.1$, and a dynamic learning rate initialized at $5e^{-5}$. We set `batch_size` $= 2$ with `gradient_accumulation_steps` $= 8$, yielding an effective batch size of 16. Unless otherwise specified, all other parameters followed the default settings of `LLaMA-Factory`.

To ensure fair comparison across training strategies, all experiments within the same group were trained under identical hyperparameters and total number of training steps. By default, we adopted an 8:1:1 split of the data into training, validation, and test sets. In cases where a validation set was not established, we instead used a 9:1 split between training and test. Early stopping was applied, terminating training if no improvement in validation scores was observed for three consecutive epochs.

3.4 Evaluation Metrics

To evaluate the quality of autoformalization models, we adopt a combination of standard NLP metrics and code-aware indicators. Our composite metric, *SMAD_Score*, is inspired by *CodeBLEU* [25], which combines n-gram similarity with syntax- and data-flow-levee analyses. While CodeBLEU is effective for general programming languages such as Python or Java, it relies on the AST parser that is not readily available for Agda. To address this, we design SMAD_Score as a lightweight alternative that captures syntax validity, surface-level accuracy, and recall-based content fidelity.

The **SMAD_Score** is defined as:

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

[1] https://github.com/GrammaticalFramework/informath/blob/main/data/datasets/

where the three components are defined as follows:

- **ERROR%**: Percentage of outputs that fail to parse in Agda, directly measuring syntactic validity and parsability.
- **BLEU-4** [15]: An n-gram precision metric up to 4-grams, rewarding exact token matches and reflecting surface-level accuracy.
- **ROUGE-1/2/L** [14]: Recall-oriented metrics. ROUGE-1/2 capture unigram and bigram coverage, while ROUGE-L measures longest common subsequence, together reflecting information recall and content preservation.

When choosing the weights, we aimed to balance the contribution of all components while giving slightly higher weight to BLEU and ERROR%. This ensures that both accuracy and parsability, which are the most critical aspects for practical autoformalization, have greater influence on the final score. By combining all three components, SMAD_Score serves as an intuitive single number that summarizes overall model performance, allowing direct comparison across different training setups. For simplicity, the following figures and tables use Score to refer to SMAD_Score.

4 Results

4.1 Multi-Language Joint Training

Building on our selected base model, Qwen2.5-7B-Instruct, we conducted experiments to evaluate the effectiveness of multi-formal-language joint training (as proposed by Jiang et al. [13]) in enhancing Agda autoformalization. We further explored whether multi-natural-language joint training could improve the model's performance in translating natural language to Agda.

We designed two experimental setups using different SMAD subsets, `joint_training` and `parallel_informath`. Both are constructed from earlier datasets and aim specifically to evaluate the impact of multilingual joint training on autoformalization. While they follow identical train/test splits, they differ in how natural language statements are aligned with formal code:

- The subset of `joint_training` used in this experiment maps each formal code snippet to a **single** natural-language sentence. Total 43,044 pairs, including 3587 unique Agda–English pairs. This subset is smaller than the full dataset reported in Table 2, as it is a filtered slice retaining only one natural-language sentence per code snippet.
- `parallel_informath` maps each formal snippet to **multiple** distinct natural-language sentences. Total 2103 Agda–English pairs, including 166 unique Agda statements.

The Agda statements used in the two datasets are disjoint. To measure the effect of multi-formal and multi-natural-language joint training, we constructed four slices of each dataset and trained a model on each one. The `full` slice contains all formal–natural language pairs, the `eng` slice contains only formal–English pairs, the `agda` slice contains only Agda–natural language pairs, and the `agda_eng` slice contains only Agda–English pairs.

The validation and test sets for the two groups of experiments are drawn separately from their respective SMAD subsets. In all cases, including the following experiments, the test sets consist exclusively of Agda-English pairs, since our focus is on assessing the model's autoformalization ability in Agda. To avoid semantic overlap between training, validation, and test sets,

we enforce a strict "disjoint-source" rule: each Dedukti entry, together with all its translated counterparts in Agda, Lean, and Rocq, as well as the associated natural-language paraphrases, is treated as a single group. Training, validation, and test splits are constructed from different groups, ensuring that no formal-informal pair in the test set shares a Dedukti origin with any pair in the training or validation sets.

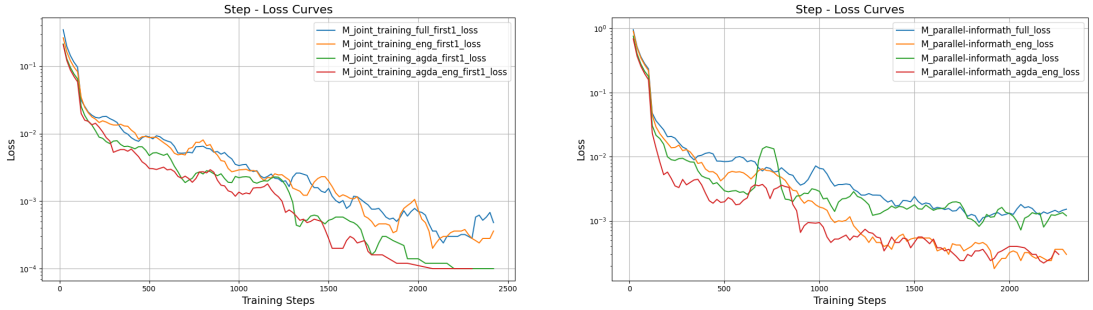
We also evaluated the Agda autoformalization abilities of the base model on both test sets. Without prompt engineering, the model tends to generate lengthy explanatory text or other irrelevant noise, which severely skews BLEU-4 and ROUGE scoring. Therefore, we designed a one-shot-style prompt that imposes explicit formatting and content constraints, along with a minimal English–Agda 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 explain,
    ↪  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 natural numbers $n$, if $n$ is odd, then 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", which any large pre-trained model already knows. 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 model's autoformalization capability is negligible. Then the output can be used as a baseline to measure the improvement of the subsequent fine-tuned models. For consistency, all test datasets for testing un-fine-tuned models follow this design principle.

Figure 2 shows the training losses and Table 3 the evaluation metrics. We observe that:

1. **Fine-tuning is powerful.** The base model performs poorly (BLEU-4 = 53.27, Syntax Err. = 90.25%), producing mostly unusable output. Fine-tuning reduces syntax errors



(a) Using `joint_training` subset. (b) Using `parallel_informath` subsets.

Figure 2: Training loss curves of models trained on different dataset slices.

Table 3: Model 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
joint_training_full	98.36	99.31 / 98.94 / 98.81	3.90	97.77
joint_training_eng	98.63	99.48 / 99.13 / 98.99	3.62	98.01
joint_training_agda	98.98	99.64 / 99.48 / 99.32	4.74	97.83
joint_training_agda_eng	98.91	99.54 / 99.26 / 99.27	5.29	97.57
parallel_informath_full	88.98	92.28 / 85.85 / 91.92	7.67	90.47
parallel_informath_eng	88.61	92.53 / 87.08 / 92.04	9.95	89.70
parallel_informath_agda	88.30	91.04 / 85.33 / 91.06	18.19	86.28
parallel_informath_agda_eng	85.71	89.62 / 85.14 / 89.53	13.04	86.86

dramatically (to below 4% in top models) and boosts composite scores by over 100%.

2. **Large sample scenarios show saturation.** When Agda–English data is sufficient, multi-language joint training offers limited gains (all BLEU-4 ≈ 99), indicating that with sufficient Agda–English examples the model already learns an almost perfect mapping. Nonetheless, 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. **Low-resource scenarios show clear benefits.** In small-data conditions, multi-formal and multi-natural joint training reduces syntax errors significantly (e.g., Syntax Err. from 13.04% to 7.67%). This suggests that parallel data in other languages and formalisms provide valuable structural signals, enhancing the model's autoformalization capabilities.

4.2 Ablation Study on Agda Data Ratios

In the previous section, we observed that parallel examples of multiple formal languages and natural languages can significantly improve performance when Agda–English samples are scarce. In this section, we further investigate the role of Agda data volume in low-resource scenarios by conducting slicing experiments with different Agda-to-Dedukti ratios, based on the `joint_training` dataset.

Although the ablation study is derived from the same dataset as the joint training experiments, the partitioning strategy is different. Here, the test set is fixed to 100 Agda–English pairs, randomly sampled from `joint_training` and kept constant across all slices. The remaining data are used to construct training sets with varying numbers of Agda and Dedukti examples.

Specifically, we defined six configurations. Slice 1 contains 500 Agda examples. Slice 2 extends this by adding 500 additional Dedukti examples. Slice 3 is a reduced version of Slice 2, consisting of only 125 Agda examples (from Slice 1) together with the same 500 Dedukti examples. Slices 4 and 5 further extend Slice 2 by incorporating 1,500 new Dedukti or Agda examples, respectively. Finally, Slice 6 represents a reversed low-resource setting, combining 500 Agda examples with only 125 Dedukti examples sampled from Slice 2.

Figure 3 shows the training losses. Table 4 shows the number of Agda and Dedukti examples in each training set and the corresponding results. We observe that:

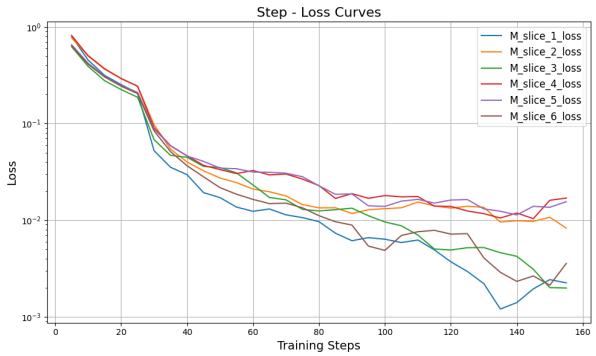


Figure 3: Training losses of slice 1–6 models.

1. **Agda data volume/number is the dominant factor.** Increasing the number of Agda statements (e.g., from 500 to 2000) consistently improves BLEU-4 scores and reduces syntax errors, confirming that more Agda data is the most effective way to enhance model performance.

2. **Dedukti provides improvement within limits.** When Agda data is sufficient, adding a small number of Dedukti samples (e.g., 125) can further improve code formatting and structural accuracy, achieving the lowest observed syntax error rate (2%). With limited Agda data, moderate Dedukti support (e.g., 500 samples) also helps reduce syntax errors and slightly improves BLEU/ROUGE. However, excessive Dedukti input (e.g., 2000 samples) may introduce noise, degrading grammatical correctness.

3. **Low-resource scenarios benefit the most.** Even with only 125 Agda samples, the model performs far better than the baseline when supplemented with 500 Dedukti examples, demonstrating that structurally similar formal languages can effectively supplement Agda learning in scarce settings.

4.3 An Improved Model

In the previous sections, 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, so the high scores clearly indicate data bias.

Table 4: Model performance in ablation study

Model	Agda	Dedukti	BLEU-4	ROUGE-1/2/L	Syntax Err.%	Score
Baseline	N/A	N/A	52.56	69.32 / 48.88 / 59.33	93	38.60
Slice 1	500	0	97.73	98.63 / 96.94 / 98.17	5	96.83
Slice 2	500	500	98.35	98.90 / 98.15 / 98.89	4	97.62
Slice 3	125	500	96.74	97.74 / 96.20 / 97.37	10	94.49
Slice 4	500	2000	98.11	98.39 / 97.23 / 98.24	7	96.27
Slice 5	2000	500	98.59	99.02 / 98.42 / 98.93	5	97.39
Slice 6	500	125	98.18	98.81 / 97.89 / 98.65	2	98.20

In Section 4.4, we show that the models do still generalize, but the real-world BLEU-4 and ROUGE scores are about 60–75.

We believe the bias can be explained as follows. The corpus consists of statements from various mathematical domains. As theorems from a given domain tend to share certain fixed patterns, the model may learn these patterns, resulting in artificially high scores on the test set. Furthermore, there is less variation than in a hand-written corpus in how these patterns are translated into natural language.

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

- The training set `final_data_training` mainly consists of simple mathematical definitions, aiming to allow the model to first master the basic grammatical knowledge of formalized languages and their mapping to mathematical terminology. These definitions cover the concepts used in the sample of "100 theorems" [28] that we used for testing, as well a few dozen other ones from algebra and set theory.

- The test set `final_data_testing` consists of manually written mathematical theorems, which are derived from the "100 theorems" corpus, evaluating the model's ability to generalize from definitions to theorem statements.

Based on the experimental results of Section 4.1 and 4.2, we adopt multi-formal and multi-natural language joint training in `final_data`, and the proportion of each formal language in the dataset is consistent, so as to maximize the automatic formalization ability of our model.

Figure 4 shows the training losses and Table 5 the evaluation metrics. We observe that:

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 syntax error rate has not increased much compared with the model `parallel_informath_full`.

   The drop in BLEU-4/ROUGE scores is acceptable, because the performance of the baseline model is considerably worse than its performance in Section 4.1. Comparing BLEU-4/ROUGE $\Delta\%$, 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 increased severely to 20.5%. We believe this is likely to be due to the model overfitting in more training steps, or learning more non-Agda expression patterns, damaging the core task performance.

Table 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
Epoch 1	76.16	89.03 / 74.94 / 83.22	7.93	83.60
Epoch 3	77.78	89.86 / 76.63 / 84.37	20.48	80.14

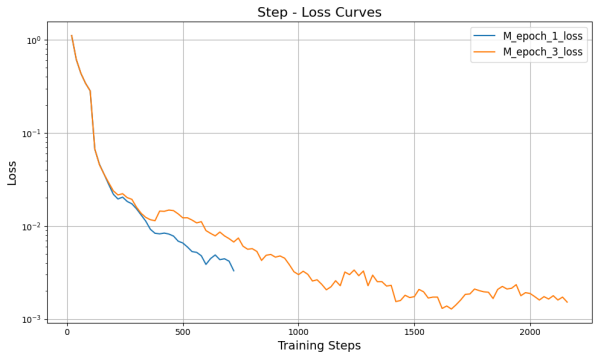


Figure 4: Training losses of fine-tuned models at different epochs.

The new training set significantly alleviates the bias problem of the previous dataset and allows the model to be trained and evaluated realistically in more challenging scenarios.

4.4 Evaluation on Human-Written Statements

To further evaluate the model's generalization ability, we constructed `natural_statements`, a dataset with human-written English statements. This set has 228 pairs in total (57 English statements with each of the 4 formalisms). They cover a subset of the "100 theorems" [28] to show variation over mathematical subfields. Table 7 shows some examples from this set. We then evaluated several representative models on these statements to test their generalization abilities. Table 6 shows the results. We see that:

- The previous model trained on `joint_training` exhibits worse generalization than before: its BLEU-4 score dropped sharply from nearly 98% to around 60-65%, and its syntax error rate increased from 3-5% to 17% on the new test set. Despite the severe degradation, it still performs noticeably better than the un-fine-tuned baseline.
- The final model shows strong generalization ability, the BLEU-4/ROUGE score does not drop sharply, and the syntax error rate (8.77%) is almost as good as the previous models.
- The sharp contrast in syntax error rates between `Epoch 1` and `Epoch 3` suggests that `Epoch 3` may overfit or be adversely affected by noise from unrelated formal languages.

Table 6: 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
joint_training_full	60.37	74.00 / 52.22 / 68.12	17.54	69.42
joint_training_agda_eng	65.00	76.66 / 55.48 / 70.90	17.54	71.91
Epoch 1	68.67	79.61 / 61.97 / 73.84	8.77	77.50
Epoch 3	65.56	78.03 / 59.66 / 70.91	22.81	70.83

Table 7: Comparison of Synthetic vs. Human-Written Statements

Agda Code	Informath Statement	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 we can prove that r is equal to the radius of c. Then we can prove 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.

4.5 Manual Evaluation

To better understand the remaining failure modes of our fine-tuned model, we examined several representative theorems and compared the output of model `Epoch 3` on human-written statements against the label. Table 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:

1. **Unsupported Infix Operators.** The model frequently emits operators like `==`, `<=`, `>` that are not defined in the 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), 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 project's 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 theorem's 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 applied a simple symbolic normalization step that replaced these operators with equivalent prefix forms or function calls compatible with standard Agda syntax. This reduced the number of syntax errors from 13 (22.8%) to just 2 (3.5%). These results indicate that many outputs are already structurally close to correctness and can be made fully functional through

lightweight symbolic post-processing. Since such transformations consist mainly of systematic replacements and syntax adjustments, they are easily mechanizable and can be integrated as a general optimization layer to enhance the overall performance of autoformalization systems.

Table 8: Selected Theorem Outputs and Expert Comments

Model Output	**Label**	**BLEU**	**Expert Comment**
`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.*
`postulate thm22 : uncountable Real`	`postulate thm22 : not (denumerable Real)`	23.45	*Expert: "Uncountable" is meaningful but not in scope.*
`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.*
`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 contains the unsupported infix operator* `==`*.*
`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* `<=`*.*

4.6 A Note on Evaluation

Autoformalization evaluation beyond string-level and syntactic methods is largely an open problem, recently addressed in [18]. As our sole automated method, we used type checking in Agda, showing that 23 out of 57 (40%) statements in the case study were type-correct after the post-processing. This is a non-trivial result in a strict type system such as Agda's, where propositions make extensive use of dependent types. The ultimate test would, of course, be not only type correctness but equivalence with the intended meaning. How to assess this is an open question; logical equivalence is obviously too weak a criterion, and something more intensional is desired.

Constructive type theory, and thereby Agda, has the concept of **definitional equality**, which means convertibility to the same normal form via a chain of definitions (including beta

conversion). This notion is more intensional than logical equivalence, but it seems to be too strong in some respects and too weak in others. For example, $2 + 2$ and 2×2 and 2^2 are definitionally equal but arguably not the same in intensional meaning; the example comes from Frege [9], as an example of expressions that have the same reference but different senses. On the other hand, $(a + b) + c$ might be considered equal to $a + (b + c)$ by associativity, but these expressions are not definitionally equal, because their equality requires a proof.

Since human evaluation is still needed to assess all aspects of autoformalization, evaluators are burdened with the task of reading and understanding complicated formulas. However, symbolic informalization can be a help even here: from any candidate formula that is syntactically correct, we can generate accurate natural language equivalents. These can be much easier for a human reader to assess than the formulas themselves.

5 Conclusions

We have presented a reproducible symbolic pipeline, based on Informath, for generating large-scale, multilingual autoformalization corpora. To showcase the utility of the pipeline, we constructed SMAD, a 400K pair multilingual autoformalization corpus covering four proof assistants (Dedukti, Agda, Rocq, Lean) and three natural languages (English, French, Swedish), all derived via the Informath to guarantee logical correctness and paraphrase diversity. Our contribution lies not only in releasing this corpus, but in introducing a systematic methodology for creating new autoformalization resources. This approach establishes a robust and extensible foundation that can be readily adapted, expanded, and replicated in future research. The SMAD corpus and the Informath software are available on GitHub[2].

By fine-tuning Qwen2.5-7B-Instruct with LoRA on data generated by Informath, we obtained strong BLEU and ROUGE scores together with competitive syntax error rates. At this stage, the model is already capable of handling student-level autoformalization tasks with high reliability, and when faced with more complex tasks it can still provide meaningful guidance. 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 four different formal languages simultaneously.

The main advantages of symbolic informalization over LLMs are its reliability (in principle) and traceability, as well as its high speed and low compute cost. The entire SMAD dataset we used took around 30 seconds of CPU time (single core) and 2.4 megabytes of memory to produce on a Macbook Air M2 from 2023. This is in contrast with [13], where the production of 332k pairs cost USD 3500 with Open AI's GPT-4. The authors report: "If we had more resources, we could further boost the diversity of the informal statements by sampling more than one informal statement for each formal statement, and could extend to more formal libraries". Such a boost can be achieved with a negligble additional cost by a symbolic method that can automatically convert Dedukti statements to hundreds of natural language variants, together with existing third-party tools that convert content from other formalisms to Dedukti.

We have not yet evaluated our results with the same data as [13], because the Informath grammar only covers a part of the vocabulary that occurs in it. This issue shows one of the bottlenecks of our approach: its performance depends on a lexicon that maps formal concepts

[2] https://github.com/GrammaticalFramework/informath

to natural language words. [21] shows how the natural language side of the data can be built by a semi-automatic way, but its relation with formal concepts must still be defined by a human, as a symbol table that maps Dedukti constants to GF constants.

Looking ahead, we intend to extend the natural statements test set, starting with all the 100 theorems of [28], which cover a wide range of mathematical contents. We are continuously extending the coverage of Informath by incorporating additional proof assistants (e.g., HOL, Isabelle, Mizar) and natural languages (e.g., Finnish, German, Spanish, Chinese), thereby broadening the applicability of our approach and further testing the effects of multilingual joint training. This includes extensions into research-level mathematics, where LLMs have less material available on the web, whereas symbolic informalization can quickly provide more data.

The experiments described here could be extended by fine-tuning more powerful LLMs such as Llama 4 and the Qwen 3 series, and by evaluating model performance on larger, manually curated benchmarks. Additionally, we could explore the fine-tuned models' capability for bidirectional translation, focusing on the informalization of formal code. In this way, we can see how much of the reliability of symbolic informalization is lost in LLMs. While some quality is expected to be lost, LLMs can be helpful in ranking symbolic informalization results in terms of fluency. Symbolic methods can guarantee that all generated texts are semantically correct, but LLMs can guide in deciding which of them sound natural.

Acknowledgments

We are grateful to Josef Urban for his valuable advice about the experiments and to Frédéric Blanqui and Rishikesh Vaishnav for their help with Dedukti. Mateja Jamnik helped us in framing the conclusions. The comments and questions from the anonymous referees were an immense help in improving all aspects of the paper. Ranta's work received support from the EuroProofNet COST action 20111.

References

[1] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory, 2023. https://arxiv.org/abs/2311.07185.

[2] Wikipedia authors. Algebra of Sets, 2025. https://en.wikipedia.org/wiki/Algebra_of_sets.

[3] Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics*, pages 261–279, Cham, 2015. Springer International Publishing.

[4] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer Verlag, 2004.

[5] Ana Bove, Peter Dybjer, and Ulf 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*, pages 73–78. Springer, 2009.

[6] Gary Chartrand, Albert Polimeni, and Ping Zhang. *Mathematical Proofs.* Pearson, 2007.

[7] Marcos Cramer, Bernhard Fisseni, Peter Koepke, Daniel Kühlwein, Bernhard Schröder, and Jip Veldman. The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts. In *CNL*, pages 170–186, 2009.

[8] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[9] Gottlob Frege. *Grundgesetze der Arithmetik*. H. Pohle, Jena,, 1893.

[10] Mohan Ganesalingam. *The Language of Mathematics: A Linguistic and Philosophical Investigation*. Springer, 2013.

[11] Thomas Hallgren and Aarne Ranta. An Extensible Proof Text Editor. In M. Parigot and A. Voronkov, editors, *LPAR-2000*, volume 1955 of *LNCS/LNAI*, pages 70–84. Springer, 2000.

[12] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[13] Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multi-language diversity benefits autoformalization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 83600–83626. Curran Associates, Inc., 2024.

[14] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, 2004. Association for Computational Linguistics.

[15] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, 2002. Association for Computational Linguistics.

[16] Andrei Paskevich. The Syntax and Semantics of The ForTheL Language, 2007.

[17] Gabrijela Perković, Antun Drobnjak, and Ivica Botički. Hallucinations in LLMs: Understanding and addressing challenges. In *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, pages 2084–2088. IEEE, 2024.

[18] Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17958–17980, Suzhou, China, November 2025. Association for Computational Linguistics.

[19] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

[20] Aarne Ranta. Translating between language and logic: What is easy and what is difficult. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 5–25, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[21] Aarne Ranta. Towards multilingual autoformalization and informalization of mathematics. In *SLTC-2024, Swedish Language Technology Conference*, 2024. https://sltc2024.github.io/abstracts/ranta.pdf.

[22] Aarne Ranta. Informath. https://github.com/grammaticalframework/informath, 2025. GitHub repository; accessed 2025-08-07.

[23] Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, and Prasanth Kolachina. Abstract Syntax as Interlingua: Scaling Up the Grammatical Framework from Controlled Languages to Robust Pipelines. *Computational Linguistics*, 46(2):425–486, 06 2020.

[24] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.

[25] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

[26] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, sep 2014.

[27] Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef 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*, CPP 2020, page 85–98, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Freek Wiedijk. Formalizing 100 theorems, 2025. https://www.cs.ru.nl/~freek/100/.

[29] Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368, 2022.