JAFI: Joint Modeling Auto-Formalization and Auto-Informalization through Training-Inference Integration

Anonymous ACL submission

Abstract

Recent advancements in large language models (LLMs) have substantially improved models' performance in auto-formalization and auto-informalization task. However, existing approaches suffer from three key limitations: (1) isolated treatment of these dual tasks despite their inherent complementarity, (2) decoupled optimization of model training and inference phases, and (3) under-explored collaboration potential among different LLMs. To address these challenges, we propose JAFI, a unified framework that integrates training and inference while jointly modeling auto-formalization and auto-informalization, through modular collaboration among specialized components. We evaluate JAFI which employs Lean 3 and Lean 4, respectively, on the mathematical dataset AMR and miniF2F. The results demonstrate that JAFI significantly surpasses existing methods across both tasks. Comprehensive ablation studies further corroborate the effectiveness of its meticulously designed modules. Additionally, JAFI's superiority is validated by its performance in the ICML 2024 Challenges on Automated Math Reasoning. Code and datasets are available at `https://anonymous.4open.science/r/JAFI-EDBC`.

1 Introduction

As a crucial component of human intelligence, the ability of coding and mathematical reasoning has attracted extensive attention in the research community of large language models (LLMs) (Shao et al., 2024; Ying et al., 2024; Guo et al., 2024; Rozière et al., 2023). Compared to the mathematical reasoning expressed through natural language (NL), i.e., informal proofs, formal languages (FL) express mathematical theorems and proofs in a machine-verifiable form akin to programming code, ensuring the reliability of the proof process (Li et al., 2024). Typical FL languages include Isabelle (Paulson and Nipkow, 1994), Coq (Huet and Paulin-Mohring, 2000), and Lean (de Moura and Ullrich, 2021), which have more rigorous syntax and logical structures compared to the more flexible NL. The tasks of **auto-formalization** and **auto-informalization** aim to convert natural language descriptions of mathematical problems into formal statements and proofs, and vice versa (Wang et al., 2018). As the forms of automated theorem proving (ATP), auto-formalization and auto-informalization play a significant role in mathematical research and education (Li et al., 2024; Jiang et al., 2023, 2022).

The development of LLMs has shed new light on the performance improvement of these two tasks, primarily due to their robust capabilities of natural language understanding and reasoning (Wu et al., 2022; Azerbayev et al., 2023a; Jiang et al., 2022). However, directly using LLMs to achieve mathematical tasks has not yielded ideal results, mainly due to (1) data bias, as mathematical language is highly specialized and comprises a small portion of LLMs' pretraining data, and (2) the inherent difficulty of the tasks, given the specialization and complexity of mathematical reasoning.

Previous efforts have attempted to enhance LLM performance on these two tasks, which can be broadly categorized into *training-based* and *inference-based* methods (Li et al., 2024). **Training-based** methods (Azerbayev et al., 2023a; Xin et al., 2024b; Azerbayev et al., 2023b; Shao et al., 2024; Ying et al., 2024) typically involve fine-tuning LLMs on extensive datasets containing both informal and formal mathematical data, thereby enhancing the model's general mathematical capabilities. On the other hand, **inference-based** methods (Jiang et al., 2022; Xin et al., 2023; Patel et al., 2023; Zhao et al., 2023) utilize techniques such as sophisticated prompt engineering and in-context learning (ICL) to directly perform auto-(in)formalization tasks with frozen LLMs. Overall, training-based methods can enhance the model's potential across diverse mathematical tasks but require substantial computational resources, whereas inference-based methods can effectively improve LLM performance on specific tasks but have performance limits.

However, previous methods have the following issues: (1) *Overlooking the synergy between the two tasks*. Previous works have primarily focused on auto-formalization (Wu et al., 2022; Jiang et al., 2022, 2023), and although some works (Wu et al., 2022; Azerbayev et al., 2023a; Lu et al., 2024) have considered auto-informalization, they seldom consider their interrelation. (2) *Lack of integration between training and inference*. Training-based and inference-based methods each have their advantages, but few works have approached these tasks from both aspects to balance training costs and model performance. (3) *Lack of research on collaboration between different models*. The auto-(in)formalization task require expertise in specific mathematical languages, translation capabilities between NL and FL, and ICL abilities. However, existing research has not explored solving these tasks through collaboration between multiple models.

We propose **JAFI**, a **J**oint framework for **A**uto-**F**ormalization and auto-**I**nformalization, that addresses these challenges through three key innovations. (1) By integrating a carefully designed *memory* module and *retrieval* module, JAFI effectively leverages the synergy between the two tasks, maximizing data utility within the tasks. (2) JAFI offers a holistic approach to model training and inference by accumulating high-confidence samples from both tasks into a unified knowledge base, thus facilitating modeling training. (3) During inference, JAFI explores the use of multiple language models for different subtasks, optimizing task completion by leveraging the distinct advantages of each LLM.

The paper's contributions are threefolds:
1. **Integrated Framework**: We propose the novel JAFI framework, which effectively tackles both auto-formalization and auto-informalization tasks, featuring a seamless training-inference loop.
2. **Architectural Innovations**: JAFI is designed with specialized modules for diverse subtasks and promotes adaptive model collaboration.
3. **Empirical Validation**: Through extensive experiments on the AMR and miniF2F mathematical datasets, JAFI demonstrates state-of-the-art performance. Our ablation studies provide further evidence of the efficacy of the modules within JAFI.

2 Related Work

Auto-formalization aims to convert theorems and proof to formal code automatically. (Wang et al., 2018) first introduce deep learning models for auto-formalization. Drawing inspiration from the sequence-to-sequence models used in neural machine translation (Sutskever et al., 2014), various encoder-decoder frameworks (Luong et al., 2017; Lample et al., 2018) are experimented with to transform mathematical texts written in LATEX into the Mizar language (Szegedy, 2020).

The development of *large language models* (LLMs) and their in-context learning capabilities (Brown et al., 2020) has created new opportunities for auto-formalization. Studies (Wu et al., 2022; Agrawal et al., 2022; Gadgil et al., 2022) explored the use of PaLM (Chowdhery et al., 2023) and Codex (Chen et al., 2021) with few-shot prompting to convert mathematical problems into formal languages like Isabelle and Lean seamlessly. Several researchers (Jiang et al., 2022; Patel et al., 2023; Zhao et al., 2023; Xin et al., 2023) propose more structured approaches for auto-formalization. For example, DSP (Jiang et al., 2022) uses Minerva (Lewkowycz et al., 2022) to generate informal proofs and transforms them into formal sketches, utilizing ATP systems to fill in the missing components of the proof sketch.

Furthermore, a body of research (Azerbayev et al., 2023a; Jiang et al., 2023; Azerbayev et al., 2023b; Shao et al., 2024; Ying et al., 2024) focuses on training LLMs with extensive datasets comprising both informal and formal mathematical data to evaluate their auto-formalization performance. Despite these advancements, existing work often lacks exploration into *the integration of training-based and inference-based methods*.

Conversely, due to the inherent inconsistencies in natural language, research into **auto-informalization** is relatively sparse (Li et al., 2024; Lu et al., 2024; Jiang et al., 2023). Overall, there is a significant gap in *exploring the synergy between auto-formalization and auto-informalization tasks*.

3 Proposed Method

3.1 Problem Formulation

Given a paired dataset $\mathcal{D} = \{(i_j, f_j)\}_{j=1}^{N}$ consisting of N aligned informal-formal proof pairs, where i_j denotes an informal proof (natural language with mathematical reasoning) and f_j rep-

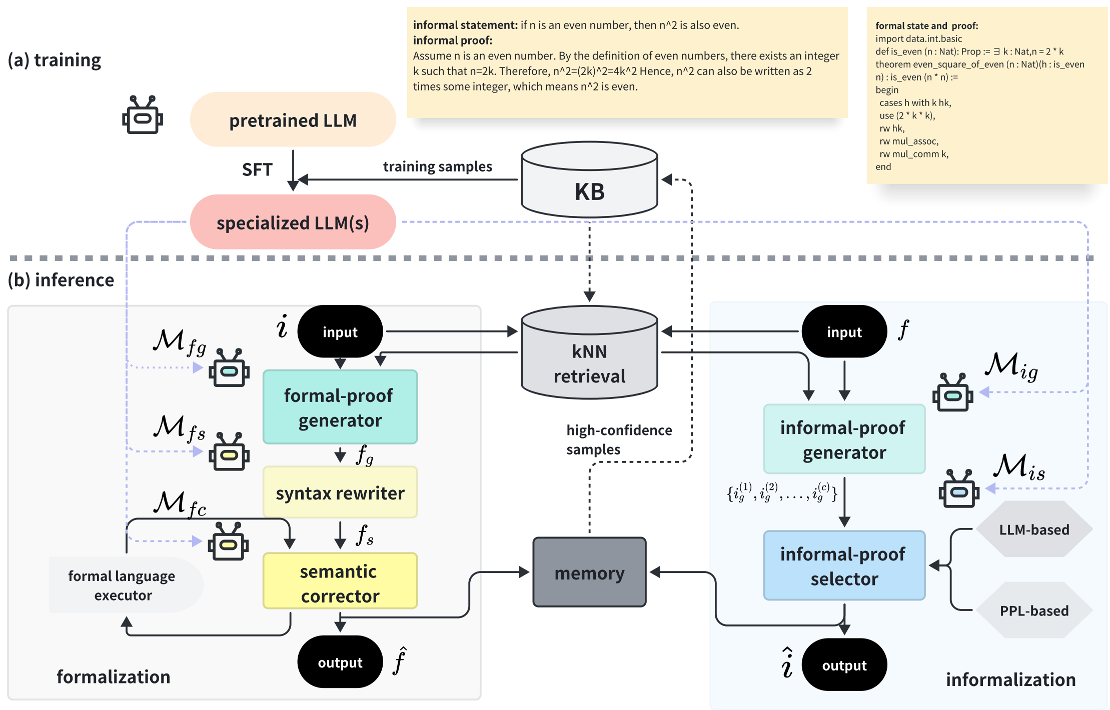

Figure 1: The overall framework of JAFI. Subfigure (a) (upper part) displays the model training process, while subfigure (b) (lower part) depicts the inference architecture. The memory stores high-confidence inference data, integrating them into a unified KB. The data is used to fine-tune specialized LLM(s), which are then applied in the submodules of the inference process.

resents its corresponding formal proof (machine-verifiable code), we formally define the two complementary translation tasks:[1]

Auto-formalization Given an informal proof i, this task focuses on generating its formal counterpart f, denoted as $i \rightarrow f$. The translation requires preserving logical equivalence while adapting to the strict syntax of formal languages.

Auto-informalization Conversely, given a formal proof f, this task aims to produce its human-readable informal version i, i.e., $f \rightarrow i$. The process involves recovering natural language explanations from symbolic representations without losing mathematical rigor.

Figure 1(a) illustrates an example of this bidirectional translation. Consider the theorem: "If n is an even number, then n^2 is also even." The informal proof (in natural language) and its formal counterpart (in a theorem prover like Lean) constitute a task pair. Auto-formalization converts the natural language proof into Lean code, while auto-informalization achieves the inverse transformation.

[1]Both i_j and f_j can be decomposed into theorem statements and their proofs. For notational simplicity, we collectively refer to them as informal/formal proofs.

3.2 Framework Overview

Figure 1 illustrates the **JAFI** framework, comprising two integrated components: the *Model Training* subsystem (upper panel a) and the *Stepwise Inference* subsystem (lower panel b). The inference subsystem contains distinct pipelines for auto-formalization (left) and auto-informalization (right).

For auto-formalization, the input informal proof i first undergoes context retrieval from the knowledge base (KB), fetching k relevant theorem-proof pairs $\{r_j\}_{j=1}^{k}$. The generator then produces candidate formal proof f_g using this context, followed by syntax correction to yield f_s. Finally, semantic correction iteratively refines f_s using formal executor feedback, producing $\hat{f}$.

Auto-informalization handles the inverse translation through a three-stage process: retrieval of similar formal proofs, generation of c candidate informal proofs $\{i_g^{(m)}\}_{m=1}^{c}$ via in-context learning, and selection of optimal output $\hat{i}$ using quality metrics (perplexity or LLM-based assessment). The asymmetric design accounts for formal proofs' syntactic rigidity versus informal proofs' natural language flexibility.

JAFI's *Memory* module bridges training and inference by collecting high-confidence predictions validated through formal verification and semantic consistency checks. These validated samples enrich the KB and enable continuous model improvement through curriculum learning, creating a self-reinforcing loop where enhanced models generate better training data, which subsequently improves model performance.

3.3 Retrieval Module

JAFI is built with a unified retrieval module to extract the most relevant samples for both tasks. Specifically, for an input q, the retrieval module $\mathcal{R}$ identifies the k samples that are most closest to q from the indexed dataset, expressed as:

$$\{r_j\}_{j=1}^{k} = \mathcal{R}(q, \mathcal{D}), \tag{1}$$

where $\mathcal{D}$ denotes the stored and indexed sample set of size N, and q represents either i in the formalization task or f in the informalization task.

Formally, in module $\mathcal{R}$, we utilize an encoder to convert the query q and the indexed dataset $\mathcal{D}$ into dense vector $\mathbf{e}_q \in \mathbb{R}^d$ and an embedding matrix $\mathbf{E} \in \mathbb{R}^{2N\times d}$. Then, the indexes of k samples are retrieved by

$$\{idx_j\}_{j=1}^{k} = \text{DenseRetriever}(\mathbf{e}_q, \mathbf{E}; k), \tag{2}$$

where the index idx_j corresponds to r_j's remapped position in the KB. The operation $\text{DenseRetriever}(;)$ can be implemented with various k-Nearest Neighbor (kNN) algorithms. More detailed settings can be found in Appendix A.

3.4 Formalization Process

As illustrated in the left part of Figure 1(b), the objective of auto-formalization process is predicting the corresponding formal proof $\hat{f}$ given informal proof i. This process mainly consists of four stages: *retrieval*, *generation*, *syntax rewriting*, and *semantic correction*. The detailed process is outlined in Algorithm 1.

At first, module $\mathcal{R}$ retrieves the k most relevant examples $\{r_j\}_{j=1}^{k}$ from the annotated dataset $\mathcal{D}$ based on the input informal statement and proof i. These examples together with i, serve as the input to the *generation* module, where the formal proof generator $\mathcal{M}_{fg}$ generates a candidate f_g. Next, in the *syntax rewriting* module, the formal language expert $\mathcal{M}_{fs}$ performs syntax corrections on f_g to produce f_s. Finally, the *semantic correction* module iteratively refines f_s based on error messages returned by the formal language executor $\mathcal{E}$, resulting in the final prediction $\hat{f}$. [2]

Algorithm 1 Formalization Process

Require: Informal proof i, training dataset $\mathcal{D}$, retrieval model $\mathcal{R}$, generator (model) $\mathcal{M}_{fg}$, syntax rewriter (model) $\mathcal{M}_{fs}$, semantic corrector (model) $\mathcal{M}_{fc}$, prompt templates $\mathcal{T}_{fg}$, $\mathcal{T}_{fs}$, $\mathcal{T}_{fc}$, formal language executor $\mathcal{E}$, parameter k
Ensure: Predicted formal proof $\hat{f}$
1: **Retrieval:** Retrieve the k samples most similar to i, i.e., $\{r_j\}_{j=1}^{k} = \mathcal{R}(i, \mathcal{D})$;
2: **Generation:** Generate candidate formal proof $f_g = \mathcal{M}_{fg}(i, \{r_j\}_{j=1}^{k}; \mathcal{T}_{fg})$;
3: **Syntax Rewriting:** Perform syntax correction on the candidate f_c to obtain $f_s = \mathcal{M}_{fs}(f_g; \mathcal{T}_{fs})$;
4: **Semantic Correction:**
5: Initialize $f_c^{(0)} = f_s$
6: **Loop:**
7: $f_c^{(j+1)} = \mathcal{M}_{fc}(f_c^{(j)}; \mathcal{T}_{fc}, \mathcal{E})$;
8: **until** formal language executor $\mathcal{E}$ returns the correct result, or maximum iterations are reached.
9: Return $f_c^{(j+1)}$ from the last iteration;

In the *semantic correction* module, f_s is used as the initial (iteration 0) corrected formal proof, denoted as $f_c^{(0)}$. It is refined iteratively by the sematnic corrctor $\mathcal{M}_{fc}$ (as described in line 7 of Algorithm 1) until the formal language executor $\mathcal{E}$ executes $f_c^{(j+1)}$ without errors or a maximum number of iterations is reached. The iterative process is formalized as

$$e^{(j)} = \mathcal{E}(f_c^{(j)}), \tag{3}$$

$$m^{(j)} = \text{EMP}(e^{(j)}), \tag{4}$$

$$f_c^{(j+1)} = \mathcal{M}_{fc}(f_c^{(j)}, m^{(j)}; \mathcal{T}_{fc}), \tag{5}$$

where $e^{(j)}$ is the raw output from executing $f_c^{(j)}$ by executor $\mathcal{E}$. Given that this output can be lengthy and complex, potentially exceeding the context length limit of the LLM, we use an error message processor EMP to simplify it into $m^{(j)}$. EMP can either be a rule-based method to remove redundant and unnecessary information or a model-based method to summarize the raw output.

Specifically, three LLMs are respectively used as $\mathcal{M}_{fg}$, $\mathcal{M}_{fs}$ and $\mathcal{M}_{fc}$, with the corresponding prompt template $\mathcal{T}_{fg}$, $\mathcal{T}_{fs}$, $\mathcal{T}_{fc}$. These LLMs can be the same or different. $\mathcal{M}_{fg}$ needs strong ICL capabilities to understand both NL and FL for language

[2] For models $\mathcal{M}_{fg}$, $\mathcal{M}_{fs}$ and $\mathcal{M}_{fc}$, we can use them in a training-free or training-train fashion, with specific prompt templates. The used templates $\mathcal{T}_{fg}$, $\mathcal{T}_{fs}$, $\mathcal{T}_{fc}$ are listed in Appendix C.

Algorithm 2 Informalization Process

Require: Formal statement and proof f, generation model $\mathcal{M}_{ig}$, selection model $\mathcal{S}$, parameters k, c

Ensure: Predicted informal statement and proof $\hat{i}$

1: **Retrieval:** Retrieve the k most similar samples to f, i.e., $\{r_j\}_{j=1}^{k} = \mathcal{R}(f, \mathcal{D})$;

2: **Generation:** Generate candidate informal proofs $\{i_g^{(1)}, i_g^{(2)}, \ldots, i_g^{(c)}\} = \mathcal{M}_{ig}(i, \{r_j\}_{j=1}^{k}; \mathcal{T}_{ig})$;

3: **Selection:** $\hat{i} = \mathcal{S}(\{i_g^{(1)}, i_g^{(2)}, \ldots, i_g^{(c)}\}; \mathcal{D})$;

- **LLM-based:** $\hat{i} = \mathcal{M}_{is}(\{i_g^{(1)}, i_g^{(2)}, \ldots, i_g^{(c)}\}; \mathcal{T}_{is})$;
- **PPL-based:** Train LM $\mathcal{M}_{ppl}$ on dataset $\mathcal{D}$, and select $\hat{i}$ as the output by calculating the perplexity of the generated candidate informal proofs;

translation. $\mathcal{M}_{fs}$ requires a deep understanding of the specific FL, such as knowing which packages to invoke for different needs. $\mathcal{M}_{fc}$ needs the ability to fix code based on feedback from the executor.

3.5 Informalization Process

The informalization process, as illustrated in the right part of Figure 1(b), is to predict the corresponding informal proof $\hat{i}$ given formal proof f, consisting of three stages: *retrieval*, *generation* and *selection*. The detailed steps are outlined in Algorithm 2.

The *retrieval* module and *generation* module (denoted as $\mathcal{M}_{ig}$) are similar to those in auto-formalization process. However, unlike $\mathcal{M}_{fg}$ generating a single candidate formal proof with a temperature of 0, $\mathcal{M}_{ig}$ samples multiple candidate informal proofs (the number is c).

In the *selection* module, the generated candidate informal proofs $\{i_g^{(1)}, i_g^{(2)}, \ldots, i_g^{(c)}\}$ are filtered through two effective selection strategies: the *LLM-based* method and the *perplexity-based* method. In the *LLM-based* method, a frozen model $\mathcal{M}_{is}$, combined with a specific template $\mathcal{T}_{is}$, is used. Notably, this method avoids the need for parameter tuning. In the *perplexity-based* method, we train a language model $\mathcal{M}_{ppl}$ (e.g., GPT-2 (Radford et al., 2019)) on the existing dataset $\mathcal{D}$ to model the language distribution of the dataset. We then calculate the perplexity of the generated candidate informal proofs and select the one with the lowest perplexity as the final prediction $\hat{i}$.

3.6 Integrating Training and Inference

To integrate model training and inference for performance improvement, as shown in Figure 1, during JAFI's inference process, high-confidence data from both tasks are saved into the memory and uniformly added into the KB. Specifically, an prediction $\hat{f}$ verified by $\mathcal{E}$ in the formalization task is paired with its input i to form a sample $\{(i, \hat{f})\}$. In the informalization task, the prediction $\hat{i}$ with a perplexity below a certain threshold is also paired with its input f to form a sample $\{(f, \hat{i})\}$.

The data in the KB is leveraged for two purposes. First, it acts as the data source for the retrieval module, encoded and stored for future auto-(in)formalization task as in-context examples. Second, the data is used for model training, fine-tuning a general pretrained LLM into specialized LLM(s), i.e., $\mathcal{M}_{fg}$, $\mathcal{M}_{fs}$, $\mathcal{M}_{fc}$ and $\mathcal{M}_{ig}$, which are employed in the modules of the inference process [3]. In practice, depending on the amount of data available and the training budget, these data can be used to train a unified multi-task model or multiple models for different subtasks.

4 Experiments

We have conducted our experiments to answer the following questions.

RQ1: Can JAFI outperform the state-of-the-art methods on the tasks of auto-formalization and auto-informalization?

RQ2: Do the devised modules/methods in JAFI contribute to the performance during the inference process?

RQ3: Is it necessary to jointly model auto-formalization and auto-informalization for better performance?

RQ4: Is it necessary to integrate model training and inference for better performance?

RQ5: How can the collaboration between different models be utilized to achieve better results?

4.1 Experimental Setup

Dataset Our experiments leverage two mathematical reasoning benchmarks. The AMR dataset, derived from MUSTARDSauce (Castro et al., 2019), contains 4,866 training samples and 500 test samples spanning elementary to Olympiad-level mathematics (including IMO problems). Each sample comprises four components: problem name, informal statement, informal proof, and corresponding Lean3-formalized proof. For cross-version validation, we additionally evaluate on the miniF2F

[3] To align with the inference process, we actually use the input-output pairs from *generation*, *syntax rewriting* and *semantic correction* modules producing correct predictions as SFT training samples, rather than only using $\{(i, \hat{f})\}$ pairs.

dataset (Zheng et al., 2021) containing 244 validation and 244 test problems in Lean4 format, focusing on algebra and number theory from AIME/AMC/IMO competitions. More detailed information can be found in Appendix B.

Evaluation Metrics For both the auto-formalization task and the auto-informalization task, the prevalent metrics of *ROUGE-L* (Lin, 2004) and *BLEU* (Papineni et al., 2002) are used to evaluate the quality of the generated texts. Moreover, *pass rate* is also used for the auto-formalization task. A generated formal proof is considered correct if it can be successfully executed by the Lean code executor and achieves a ROUGE-L score above a specified threshold when compared with the annotated solution.

Lean Execution Environment To support the *semantic correction* module and calculation of pass rate, we configured a local Lean 3 and Lean 4 execution environment running on a Linux server with 750GB of RAM and 96 CPU cores.

LLMs in JAFI JAFI leverages DeepSeek-Coder (33B) (Guo et al., 2024) as its backbone model. For the selection module of informalization, we fine-tuned GPT-2 (Radford et al., 2019) to calculate perplexity. In our model cooperation experiments, we also evaluated proprietary LLMs including GPT-4 variants (Turbo/4o) and the Gemini series for comparison. More detailed training settings can be found in Appendix B.

Baselines Comparisons span two methodological categories: Training-free approaches include *ICL* (Wu et al., 2022) using vanilla few-shot prompting, and *ICL-retrieval* (Azerbayev et al., 2023a) augmenting prompts with mathlib knowledge base retrievals. Training-based competitors comprise *proofGPT* (Azerbayev et al., 2023a) (1.3B parameters via Codex-002 distillation) and *Llemma* (Azerbayev et al., 2023b) (7B/34B models continually pretrained on Proof-Pile-2). More detailed information can be found in Appendix B.

4.2 Overall Performance

For RQ1, we compared the performance of JAFI and the baselines on the AMR and miniF2F datasets. The experimental results are shown in Table 1. For *ICL* and *ICL-retrieval*, we used GPT-4o as the backbone and followed the inference methods described in their published papers. For *proofGPT* and *Llemma*, we downloaded and utilized their pretrained models to complete the auto-(in)formalization task.

From the results, we can observe that: 1) On the AMR dataset, JAFI achieved superior performance in both the auto-formalization and auto-informalization tasks. 2) Furthermore, our model also demonstrated the best performance in the auto-formalization task on the miniF2F dataset, thereby validating the generalizability of the JAFI approach.

4.3 Ablation Studies

In response to RQ2, we conducted a series of ablation studies to investigate the effectiveness of the proposed modules.

For the auto-formalization task, we validated the effectiveness of the *retrieval*, *syntax rewriting*, and *semantic correction* module, which are denoted as *rt*, *sr* and *sc*, respectively. The results in Table 2 show that removing any module in JAFI results in a performance drop, demonstrating the importance of incorporating the in-context samples (retrieved samples) before generation and post-processing the generated results. An quantitative case study can be found in Appendix D.

Furthermore, we randomly selected 50 samples and counted the number of correct results (samples) generated by the *generation* module (denoted as *#gen*), and the number of correct results supplemented by the *syntax rewriting* module (denoted as *#sr*). In addition, the number of correct samples supplemented by the first, second, and third attempts (iterations) of *semantic correction* are denoted as *#sc1*, *#sc2*, and *#sc3*, respectively. The results are shown in Table 3, where EMP indicates whether the FL executor's error messages were processed. According to the results, (1) both the *syntax rewriting* and *semantic correction* module can correct a significant number of samples, validating their necessity. (2) Compared to the approach without EMP, EMP allowed the model to correct more samples, demonstrating the necessity of *error message processor*.

For the auto-informalization task, we analyzed the two methods of candidate informal proofs. For the *perplexity-based* method, we trained two models, GPT-2 and Llama-7B. For the *LLM-based* method, we used Gemini-Chat, DeepSeek-V2 and GPT-4o. Additionally, a random selection method was also considered as a baseline. The experimental results in Table 4 support the following conclu-

Method	AMR-Formalization			AMR-Informalization		miniF2F-valid	miniF2F-test
	ROUGE-L	BLEU	passrate	ROUGE-L	BLEU	passrate	passrate
ICL	0.2102	0.0772	0.18	0.2102	0.0523	0.02	0.02
ICL-retrieval	0.3487	0.1000	0.46	0.2753	0.0980	0.05	0.03
proofGPT	0.2545	0.0704	0.02	0.1805	0.0472	0.11	0.08
Llemma-7B	0.2766	0.0714	0.04	0.2102	0.0482	0.24	0.26
Llemma-34B	0.3246	0.1023	0.16	0.2645	0.0743	0.28	0.25
JAFI	**0.4217**	**0.1407**	**0.78**	**0.3412**	**0.1168**	**0.45**	**0.42**

Table 1: Overall performance on the two tasks of all compared methods.

Variant	ROUGE-L	BLEU	passrate
w/o *rt*	0.2866	0.0724	0.06
w/o *sr&sc*	0.3487	0.1000	0.46
w/o *sr*	0.3642	0.1109	0.56
w/o *sc*	0.4152	0.1224	0.64
JAFI	**0.4217**	**0.1407**	**0.78**

Table 2: Ablation study of JAFI's modules on auto-formalization.

Model	EMP	*#gen*	*#sr*	*#sc1*	*#sc2*	*#sc3*
Gemini-Base	✓	21	+2	+3	+1	0
	×	21	+3	+4	+2	+1
DeepSeek-Coder	✓	29	+3	+2	0	0
	×	30	+3	+3	+1	0

Table 3: The number of correct results (supplemental) after the processing of JAFI's modules on auto-formalization.

Method	Model	ROUGE-L	BLEU
Random	-	0.2866	0.0724
Perplexity-based	GPT-2	0.3306	0.1124
	Llama-7B	**0.3350**	**0.1207**
LLM-based	Gemini-Chat	0.3198	0.0974
	DeepSeek-V2	0.3206	0.1042
	GPT-4o	0.3281	0.0989

Table 4: The effects of selecting candidate informal proofs on auto-informalization.

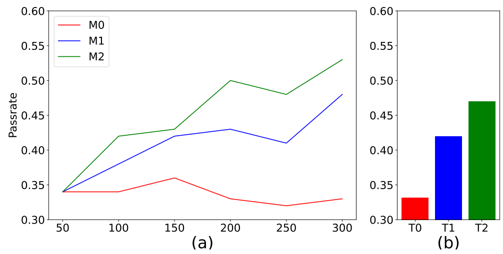

Figure 2: Auto-formalization performance of JAFI's variants under limited annotation resources. In Subfigure (a), *M0*, *M1* and *M2* represent different inference scenarios. In Subfigure (b) *T0*, *T1* and *T2* indicate different training strategies.

sions. (1) Both methods outperform the random method, indicating the importance of sample selection. (2) The *perplexity-based* method has better results, justifying the effectiveness of aligning language models with the language features of the training samples, even though the models are of small size.

4.4 Impacts of Joint Modeling and Integrating Training and Inference

In response to RQ3 and RQ4, we conducted specific experiments on the auto-formalization task in a low-resource environment, in which only 100 annotated samples randomly selected from the training set were considered. Then, we recorded the average pass rates of JAFI's variants for 300 randomly selected test samples of auto-formalization, where the used model is DeepSeek-Coder.

To investigate the necessity of **jointly modeling the two tasks**, we proposed three variants of JAFI, denoted as *M0*, *M1* and *M2*. These three variants achieved the auto-formalization task by six gradual steps, in each of which 50 samples were inferred. Specifically, *M0* is the variant without the *memory* module, indicating it can only retrieve from the 100 annotated samples. In each inference step of *M1*, the inferred high-confidence samples were saved to the memory, which can be used in the *retrieval* module of the next inference steps (for the rest test samples). In *M2*, besides the inferred high-confidence samples were used as *M2*, other high-confidence samples obtained by inferring 50 randomly selected samples of auto-informalization were also included into the memory, implying that it leverages the synergy between the two tasks. The pass rates of three variants for the six groups of auto-formalization samples (each group has 50 samples) are depicted in Figure 2(a). It shows *M1*'s performance improvement over *M0*, demonstrating that accumulating experiences through the *memory*

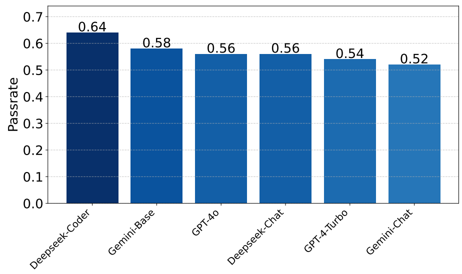

Figure 3: The performance of JAFI using a single LLM.

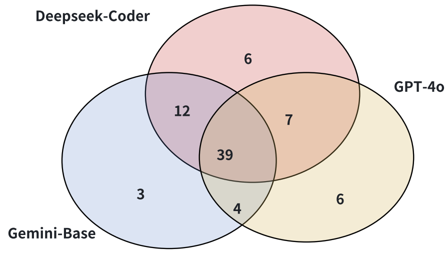

Figure 4: Number of the samples passed by three models on 100 test samples of auto-formalization.

Model/Strategy	ROUGE-L	BLEU	passrate
DeepSeek-Coder	0.4615	0.1365	0.68
Gemini-Chat	0.4082	0.1078	0.56
GPT-4o	0.4512	0.1308	0.54
MM-single	0.4306	0.1375	0.76
MM-cross	**0.4822**	**0.1412**	**0.82**

Table 5: The performance of using single LLM and the strategies of model cooperation.

module effectively enhances the model's capability. In addition, *M2*'s superiority over *M1* is attributed to jointly modeling the two tasks (leveraging the synergy between the two tasks).

To investigate whether it is necessary to **integrate model training and inference**, we proposed other three variants without the *memory* module, denoted *T0*, *T1* and *T2*. We used DeepSeek-Coder as *T0*, to infer the 300 test samples of auto-formalization directly. In *T1*, only the generator $\mathcal{M}_{fg}$ was trained by the 100 training samples. In *T2*, besides $\mathcal{M}_{fg}$, model $\mathcal{M}_{fs}$ and $\mathcal{M}_{fc}$ were also trained simultaneously with some samples which come from the 100 training samples and were validated by the FL executor. These three variants' pass rates on the 300 test samples of auto-formalization are shown as the columns in Figure 2(b). It is evident that *T1* shows significant improvement over *T0*, and *T2* further improves upon *T1*, validating the necessity of integrating model training and inference.

4.5 Impacts of Model Cooperation

To answer RQ5, we first compared the auto-formalization performance of using the same LLM in all modules of JAFI, which was not trained[4], of which the results are displayed in Figure 3. It shows that taking different LLMs as the backbone of JAFI exhibits varying performance.

Furthermore, we plotted a Venn diagram (Figure 4) to illustrate the number of the samples successfully processed by the JAFI only using Deepseek-Coder, Gemini-Base or GPT-4o when inferring 100 test samples of auto-formalization. The diagram reveals that the sample set each model can solve are distinct. Although Deepseek-Coder demonstrates the strongest overall performance, there are still 6 and 3 problems (samples) that only GPT-4o and Gemini-Base can solve, respectively.

This observation inspires us to achieve better performance through integrating the multiple models in the modules. We proposed two simple strategies for model cooperation as follows. (1) In *MM-single*, each model independently attempts to solve the problem and the first successful result passing the FL executor is returned. (2) In *MM-cross*, each of the three models acts as a *generator* to produce candidate proofs, and the mathematically stronger model Deepseek-Coder handles the *syntax rewriting* and *semantic correction*.

As shown in Table 5, the results indicate that both strategies of model cooperation outperform any single model used in JAFI, and MM-cross outperforms MM-single, validating the effectiveness of model cooperation.

5 Conclusion

In this paper, we present the **JAFI** framework, a comprehensive solution tailored for both auto-formalization and auto-informalization tasks. This framework is underpinned by carefully designed modules: *retrieval*, *syntax rewriting*, and *semantic correction* for auto-formalization, alongside a *selector* for auto-informalization. Furthermore, JAFI incorporates an innovative *memory* module, which not only records and utilizes successful past operations for future tasks but also enriches the dataset, thereby enhancing model training. Our extensive experiments, conducted using the AMR and miniF2F datasets for rigorous validation, confirm the effectiveness and robustness of JAFI and its constituent modules in advancing both tasks.

[4]The results of auto-informalization in Appendix E.

6 Limitations

This study presents two primary limitations: 1) **Insufficient model training**: Due to time and cost constraints, we trained the 33B DeepSeek-Coder model only on the AMR dataset, resulting in a relatively small amount of training data. It is essential to explore methods for constructing more comprehensive training datasets for both formalization and informalization tasks, and to study the impact of scaling up training data on model performance. 2) **Lack of exploration of alternative test-time compute methods**: Our approach predominantly focused on validating the efficacy of joint modeling of the two tasks and integration of model training and inference, ignoring other test-time compute techniques that could potentially enhance auto-formalization outcomes, such as Monte-Carlo Tree Search (MCTS) (Coulom, 2006; Xin et al., 2024a). Furthermore, recent developments include models that have strengthened reasoning capabilities during inference, such as OpenAI's o1 (Zhong et al., 2024) and DeepSeek-R1 (Guo et al., 2025). Intuitively, their enhanced reasoning abilities could improve performance in both auto-formalization and auto-informalization tasks, warranting further investigation in future work.

References

Ayush Agrawal, Siddhartha Gadgil, Navin Goyal, Ashvni Narayanan, and Anand Tadipatri. 2022. Towards a mathematics formalisation assistant using large language models. *arXiv preprint arXiv:2211.07524*.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir R. Radev, and Jeremy Avigad. 2023a. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *ArXiv*, abs/2302.12433.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023b. Llemma: An open language model for mathematics. *ArXiv*, abs/2310.10631.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Santiago Castro, Devamanyu Hazarika, Verónica Pérez-Rosas, Roger Zimmermann, Rada Mihalcea, and Soujanya Poria. 2019. Towards multimodal sarcasm detection (an _obviously_ perfect paper). *arXiv preprint arXiv:1906.01815*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Leonardo Mendonça de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *CADE*.

Siddhartha Gadgil, Anand Rao Tadipatri, Ayush Agrawal, Ashvni Narayanan, and Navin Goyal. 2022. Towards automating formalisation of theorem statements using large language models. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on MATH-AI*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *ArXiv*, abs/2401.14196.

Gérard P. Huet and Christine Paulin-Mohring. 2000. The coq proof assistant reference manual.

Albert Q Jiang, Wenda Li, and Mateja Jamnik. 2023. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *ArXiv*, abs/2210.12283.

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Phrase-based & neural unsupervised machine translation. *arXiv preprint arXiv:1804.07755*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. 2024. A survey on deep learning for theorem proving. *ArXiv*, abs/2404.09939.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Jianqiao Lu, Zhengying Liu, Yingjia Wan, Yinya Huang, Haiming Wang, Zhicheng YANG, Jing Tang, and Zhijiang Guo. 2024. Process-driven autoformalization in lean 4. *ArXiv*, abs/2406.01940.

Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. 2017. Neural machine translation (seq2seq) tutorial.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Nilay Patel, Jeffrey Flanigan, and Rahul Saha. 2023. A new approach towards autoformalization. *arXiv preprint arXiv:2310.07957*.

Lawrence Charles Paulson and Tobias Nipkow. 1994. Isabelle: A generic theorem prover.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, I. Evtimov, Joanna Bitton, Manish P Bhatt, Cristian Cantón Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre D'efossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Christian Szegedy. 2020. A promising path towards autoformalization and general artificial intelligence. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings 13*, pages 3–20. Springer.

Qingxiang Wang, C. Kaliszyk, and Josef Urban. 2018. First experiments with neural translation of informal to formal mathematics. In *International Conference on Intelligent Computer Mathematics*.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *ArXiv*, abs/2205.12615.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. 2024a. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*.

Huajian Xin, Haiming Wang, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. 2023. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*.

Huajian Xin, Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024b. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *ArXiv*, abs/2405.14333.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, et al. 2024. Internlm-math: Open math large language models toward verifiable reasoning. *arXiv preprint arXiv:2402.06332*.

Xueliang Zhao, Wenda Li, and Lingpeng Kong. 2023. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*.

Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun Lyu, Peng Shu, Xiaowei Yu, et al. 2024. Evaluation of openai o1: Opportunities and challenges of agi. *arXiv preprint arXiv:2409.18486*.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes)
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes)
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (yes)

Does this paper make theoretical contributions? (no)

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no)
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no)
- Proofs of all novel claims are included. (yes/-partial/no)
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no)
- Appropriate citations to theoretical tools used are given. (yes/partial/no)
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA)
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA)

Does this paper rely on one or more datasets? (yes)

If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets (yes)
- All novel datasets introduced in this paper are included in a data appendix. (NA)
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes)
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing. (NA)

Does this paper include computational experiments? (yes)

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (yes).
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes)
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes)
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (NA)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes)
- This paper states the number of algorithm runs used to compute each reported result. (yes)

- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes)

- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes)

- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (NA)

- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (NA)

Appendices

A Detailed Model Description

Retrieval Settings Given the substantial differences between natural language (NL) and formal language (FL), it is impractical to directly incorporate them into the retrieval module without preprocessing, as this could cause interference between the two languages. Therefore, we employ a symmetric encoding approach to manage both retrieval tasks effectively. Specifically, for the existing samples (i_j, f_j), we prepend the prefixes `Natural language statement and proof:` and `Formal statement and proof:` to i_j and f_j, respectively.

B Detailed Experiment Settings

Dataset The **AMR** dataset can be found at https://sites.google.com/view/ai4mathworkshopicml2024/challenges. It includes 4,866 samples for training and 500 samples for evaluation. Each sample in the dataset contains four fields: *name*, *informal statement*, *informal proof*, and *formal proof*, as illustrated in Figure 1(a).

The **miniF2F** dataset was first presented by OpenAI (Zheng et al., 2021) in the format of Lean 3 (https://github.com/openai/miniF2F). Later researchers converted it to an equivalent Lean 4 version (https://github.com/yangky11/miniF2F-lean4). To validate the generalizability of our method, we conducted experiments using the Lean 4 version.

Evaluation Metrics We utilized the `rouge_score` (https://github.com/google-research/google-research/tree/master/rouge) and `nltk` (https://github.com/nltk/nltk) packages for the implementation of our evaluation metrics.

LLM training settings For JAFI model, we used 33B DeepSeek-Coder as the backbone, training $\mathcal{M}_{fg}$ and $\mathcal{M}_{ig}$ with labelled data, while training $\mathcal{M}_{fs}$ and $\mathcal{M}_{fc}$ on the high-confidence data inferred from DeepSeek-Coder on training data. To ensure a fair comparison with other methods, we used a single backbone for multi-task training across these four sub-tasks. We employed a GPT-2 model for candidate selection trained on dataset with the LM object.

For model training, we combined data from all four sub-tasks, using different prompts, to train a single 33B model. The final version of our model was trained for approximately three hours on eight A100 GPUs.

Baselines We compared JAFI with the following state-of-the-art methods. Based on whether the models are trained on specialized mathematical data, they can be categorized into *training-free* and *training-based* methods. For *training-free* methods, they enhance the performance on auto-(in)formalization task through inference, including:

- *ICL* method (Wu et al., 2022) employs few-shot learning by leveraging the ICL capabilities of LLMs.
- *ICL-retrieval* method (Azerbayev et al., 2023a) enhances few-shot learning through incorporating the retrieved k-nearest neighbors from formal statements, utilizing a KB from Lean's package `mathlib`.

For *training-based* methods, they generally improve the model's overall mathematical capabilities through training on large-scale mathematical data, including:

- *proofGPT* (Azerbayev et al., 2023a) utilizes the distilled backtranslation and employs Davinci-codex-002 as the teacher model to train a student model with 1.3 billion parameters.
- *Llemma* (Azerbayev et al., 2023b) is an LLM continuously pretrained on a large-scale dataset named Proof-Pile-2 from Code-Llama(Rozière et al., 2023), available in the variants of 7B and 34B versions.

C Prompts

C.1 Prompt for Formal-Proof Generator

```
You are a math expert and familiar with
Lean 3 formal language.
Now please translate the following
statement and solution of a math word
problem into Lean 3 formal solution.
Please note that the informal solution
and the formal solution need to be
identical.

{samples}

## Problem:
{informal_statement}
```

```
## Informal Solution:
{informal_proof}

## Formal Solution in Lean 3:
```

Listing 1: Prompt for Formal-Proof Generator

C.2 Prompt for Formal-Proof Syntax Rewriter

```
You are an expert in the Lean 3 language
.
Please check the Lean code below, and if
 there are any issues, please correct
them to make it a valid, runnable code.
Note:

1. When working with mathematical
structures that cannot be effectively
computed, such as real numbers or
infinite sets, don't forget to add the
keyword  'noncomputable'.
For example:
noncomputable def inv (x : $\mathbb{R}$)
 : $\mathbb{R}$ := 1 / x

2. Pay attention to the completeness of
the code, for example, ensuring there is
 an `end` corresponding to each `begin`.

3. Always check the 'State' of the
theorom in proving, avoid unnecessary
tactics .

4. If the problem involves substitution
calculations with unknowns, carefully
choose one of [rw], [simp], or [norm_num
].

5. In one problem, put all 'import' at
the beginning of the code.

6. remember to use "#eval" to give the
final answer if the problem has a
definit output

Below are some reference Lean codes:

{samples}

For the problem "{informal_statement}",
here is a piece of code addressing this
problem:

```lean
{code}
```

Please provide your corrected code to
ensure it can run correctly, only give
the lean code:
```

Listing 2: Prompt for Formal-Proof Syntax Rewriter

C.3 Prompt for Formal-Proof Semantic Corrector

```
You are a math expert and familar with
Lean 3 formal language.
Please check the Lean code below. The
error message from the Lean 3 server has
 been given. Please correct them to make
 it a valid, runnable code.
For the problem "{informal_statement}"
Here is a piece of code addressing this
problem:
```lean
{code}

Error message:

{err_msg}

Please provide your corrected code to
ensure it can run correctly, only give
the lean code:
```

Listing 3: Prompt for Formal-Proof Semantic Corrector

C.4 Prompt for Informal-Proof Generator

```
You are a math expert and familar with
Lean 3 formal language.
Now please translate the following Lean
3 code into natural language.

You should output the natural language
statement of the problem and the natural
 language solution of the problem in the
 form of JSON. e.g. {{"Problem": xxx ,"
Solution": xxx}}

{samples}

## Formal Solution in Lean 3:

```lean
{formal_proof}
```

Problem and Solution:
```

Listing 4: Prompt for Informal-Proof Generator

## D Qualitative Analysis Case

Below, we present a simplified example to highlight the roles of the *generator, rewriter, and corrector* modules:

1) The informal problem is "John had 1/2 of a pizza and he ate 1/4 of it. How much pizza does he have left?" The ground truth formal proof is:

```
import data.real.basic
noncomputable def half : ℝ := 1/2
noncomputable def quarter : ℝ := 1/4
theorem john_pizza : half - quarter =
quarter :=
begin
    unfold half quarter, -- This
    replaces 'half' and 'quarter' with
    their definitions.
```
```

```
    norm_num, -- This performs the
    subtraction.
end
```

2) The *generator* produces the following result:

```
def john_pizza_initial : ℝ := 1/2
def john_pizza_eaten : ℝ := 1/4
def john_pizza_left : ℝ :=
john_pizza_initial - john_pizza_eaten
#eval john_pizza_left
```

3) The *rewriter* identifies two issues in the generated code: the lack of an `import` statement and the absence of the `noncomputable` modifier (in Lean, the real number type ℝ cannot directly compute concrete values).

```
import data.real.basic
noncomputable def john_pizza_initial : ℝ
 := 1/2
noncomputable def john_pizza_eaten : ℝ
:= 1/4
noncomputable def john_pizza_left : ℝ :=
 john_pizza_initial - john_pizza_eaten
#eval john_pizza_left
```

4) After the Lean executor runs the code, it returns an error: `error: code generation failed, VM does not have code for 'john_pizza_left.'` The *corrector* then modifies the code to yield the correct proof:

```
...
example : john_pizza_left = 1/4 :=
begin
  unfold john_pizza_left
  john_pizza_initial john_pizza_eaten,
  norm_num,
end
```

E Detailed Comparison Results for Different Backbone Models

Here we provide the detailed comparison results for different backbone models in auto-informalization. The BLEU and ROUGE-L scores of different models are shown in Figure 5 and Figure 6. The BLEU and ROUGE-L scores' distribution of different models over 50 samples are shown in Figure 7 and Figure 8.

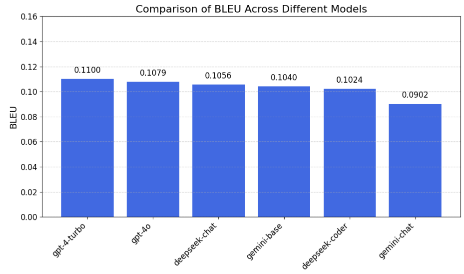

Figure 5: The BLEU scores of different models in auto-informalization.

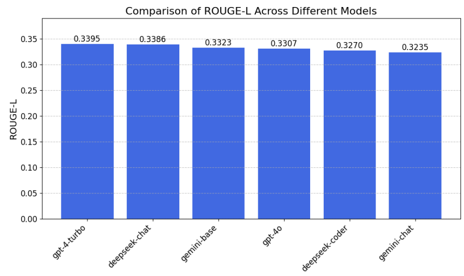

Figure 6: The ROUGE-L scores of different models in auto-informalization.

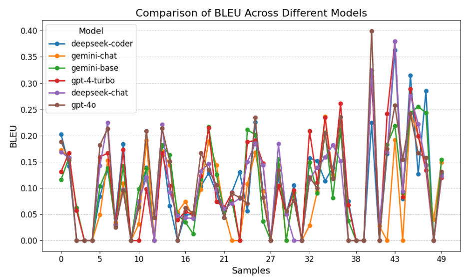

Figure 7: The BLEU scores' distribution of different models over 50 samples in auto-informalization.

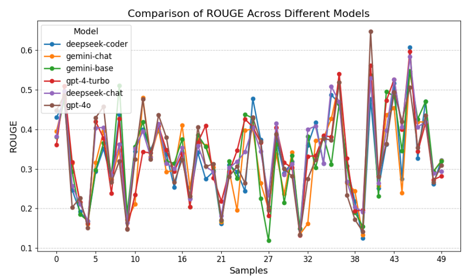

Figure 8: The ROUGE-L scores' distribution of different models over 50 samples in auto-informalization.