Introduction & Motivation

Mathematical writing and theorems have expanded quickly over the last decade. Websites such as arXiv add thousands of new theorems each year. Most of these theorems are written for human readers, expressed as a mix of natural language and LaTeX. While this format is precise for mathematicians, it is poorly suited to automatic indexing, comparison, reuse, or even undergraduate math students.

This project focuses on the gap between expert human written mathematics to machine readable representations and readability for newer mathematicians. The core problem addressed here is how to extract a theorem directly from its statement.

The goal is to convert raw theorem statements into a structured JSON object. In this project, “structured” means separating the statement into a small set of fields: a type label such as theorem or lemma, a list of assumptions and a conclusion. For a theorem like, “For an elliptic differential operator on a compact manifold, its analytical index equals its topological index.” The output would be:

```
 Parsed JSON:
{
  "type": "theorem",
  "id": "thm:elliptic_index_equality",
  "name": "Index Theorem for Elliptic Operators",
  "assumptions": [
    "Operator is elliptic",
    "Manifold is compact"
  ],
  "conclusion": "Analytical index equals topological index"
}
```

As seen above the intent is not to formalize proofs or theorems, it is to capture the components of a claim in a consistent way.

This project was done using a small dataset of theorem statements paired with structured JSON annotations. Mistral-7B-Instruct was adapted using QLoRA rather than full fine-tuning. Prompts and loss masking were designed to bias the model toward emitting only the desired JSON structure. The resulting system was evaluated through qualitative inspection, with particular attention to common failure modes.

Dataset

The dataset consists of statements like mathematical theorems or lemmas. Some examples are simplified or to reduce noise. The emphasis is on clean statements where assumptions and conclusions can reasonably be separated.

Each example is annotated according to a fixed JSON schema, an example is shown above in the previous section. The schema includes a type field, an id, a name, a list of assumptions, and a conclusion string. Assumptions are defined as conditions under which need to be true for the statement to hold. The conclusion records the main claim and excludes explanatory text.

The dataset was acquired through an Openai api prompt. The api was given various different math topics to ensure no repeat theorems or lemmas. 300 different examples were constructed and given from the api.

Methodology

The base model used is Mistral-7B-Instruct-v0.2. It was selected for its instruction-following behavior, open weights, and solid performance on reasoning tasks relative to its size.

Prompts follow a fixed structure. An instruction header specifies the expected JSON fields which were mentioned before, followed by the theorem text, and then explicit <json> delimiters.

Training uses a standard causal language modeling objective. The prompt tokens are excluded from the loss, so the model is trained only on the JSON it is supposed to produce. This keeps training focused on generating the structured output.

QLoRA is used to fine-tune the model. I used QLoRA so the model can be fine-tuned without retraining all of its weights. With QLoRA the main model stays fixed while only a small set of adapter weights is learned. This keeps memory use and training time low, but still lets the model adjust to the structured extraction task. I used google colab with T4 High Ram to train the model. Training is carried out in the Google Colab environment, with batch size, learning rate set according to available GPU memory.

Inference & Deployment Setup

At inference time, when the model is used to produce outputs for new inputs, the base model is loaded once and kept fixed. The QLoRA adapters are then applied on top of it. These adapters are small sets of learned weights that change how the model responds, but they do not trigger any further training. The model simply uses what it has already learned. Prompts are built in the same format as during training so the model sees a familiar structure.

Keeping the adapters separate from the base model is convenient in practice. The adapters take up very little storage, so they are easy to save and share. More than one adapter can also be used with the same base model, allowing the model to switch between tasks without copying or retraining the full network.

After generation, the output is processed to extract the structured result. The system looks for the first JSON block found between the chosen delimiters.

Evaluation & Results

The evaluation is mainly qualitative. It is hard to define a single numeric metric because a theorem can often be parsed in more than one reasonable way. Two different JSON representations may both be acceptable, even though they do not match exactly. For that reason, the model's output is examined directly rather than scored with a strict numerical measure.

I tested the model on a variety of theorem statements. These examples differ in length, mathematical area, and sentence structure. Some are short and direct, while others include longer

explanations or more complex phrasing. This range helps reveal how the model behaves under different conditions.

In many cases, especially for short theorems, the model produces a clear separation between assumptions and conclusions. The resulting JSON follows the expected format and captures the main claim accurately. These examples show that the model can handle straightforward statements reliably.

When errors occur, they usually fall into a few common patterns. Sometimes the model generates more than one JSON object in a single response. In other cases, it adds extra conclusions or reformulates simple corollaries that were not part of the original statement. Formatting problems also appear from time to time, and they become more common as the input gets longer.

The most difficult cases are theorems with many assumptions or multiple conclusions packed into one sentence, in other words more complicated theorems. These examples highlight the limits of instruction following when the underlying language is ambiguous.

Limitations & Ethical Considerations

There are several limitations to this work. One major constraint is hardware and available resources. Because training was done on limited compute, the model could not be trained for as long or on as much data as desired. This limits how much the model could improve during fine-tuning.

Another limitation is the input scope. The model is built to handle one theorem at a time. It cannot process multiple theorems at once, references to earlier results, or statements that

depend on long context. This makes it better suited for standalone theorems than for full mathematical sections.

The largest limitation comes from mathematical language itself. Theorems often include hidden assumptions or rely on shared understanding that is not written explicitly. Different readers may also disagree on what should count as an assumption versus the conclusion. Because of this, there is no guaranteed way to confirm that the produced structure is always correct. The output should be seen as a reasonable interpretation which there can be many of rather than a guaranteed fact.

Conclusion

This work shows that a general instruction-tuned model can be adapted to extract structured information from theorem statements using parameter-efficient fine-tuning. By training only a small set of adapter weights and keeping the base model fixed, the system learns to produce consistent JSON outputs without requiring large amounts of compute. The overall setup is relatively simple, can run on limited hardware, and can be repeated by others with similar resources.

There are several clear directions for future work. One improvement would be to expand the dataset with more theorems from different areas of mathematics and with more varied writing styles. Another direction is to enforce the output format more strictly, for example by using schema-based decoding to reduce formatting errors. Finally, the structured theorem representations could be connected to other tools, such as proof assistants to support search, reuse, and further automated reasoning.