ConvFinQA

# Introduction

**Retriever side**

* **Main.py** (in finqanet\_retriever/) → trains the retriever model.
* **Test.py** → runs the retriever to produce predictions (retriever\_outputs.json).

**2. Generator side**

The generator doesn’t directly eat the retriever’s raw JSON.  
It expects structured input with question + retrieved evidence in a specific format.

* **Convert.py** (in finqanet\_generator/)
  + A preprocessing script.
  + Reads the retriever’s output (retriever\_outputs.json).
  + Transforms it into a generator-ready dataset (e.g., generator\_input.json).
  + This is a **one-time transformation step**, not training.
* **Main.py** (in finqanet\_generator/)
  + This is the **training entry point** for the generator model (seq2seq).
  + It takes the converted data (generator\_input.json) as --train\_file, --dev\_file, etc.
  + Trains the model to generate reasoning steps and answers.
* **Test.py** (in finqanet\_generator/)
  + Runs inference with a trained generator model.
  + Produces final predicted answers.

# 0. Train the retriever

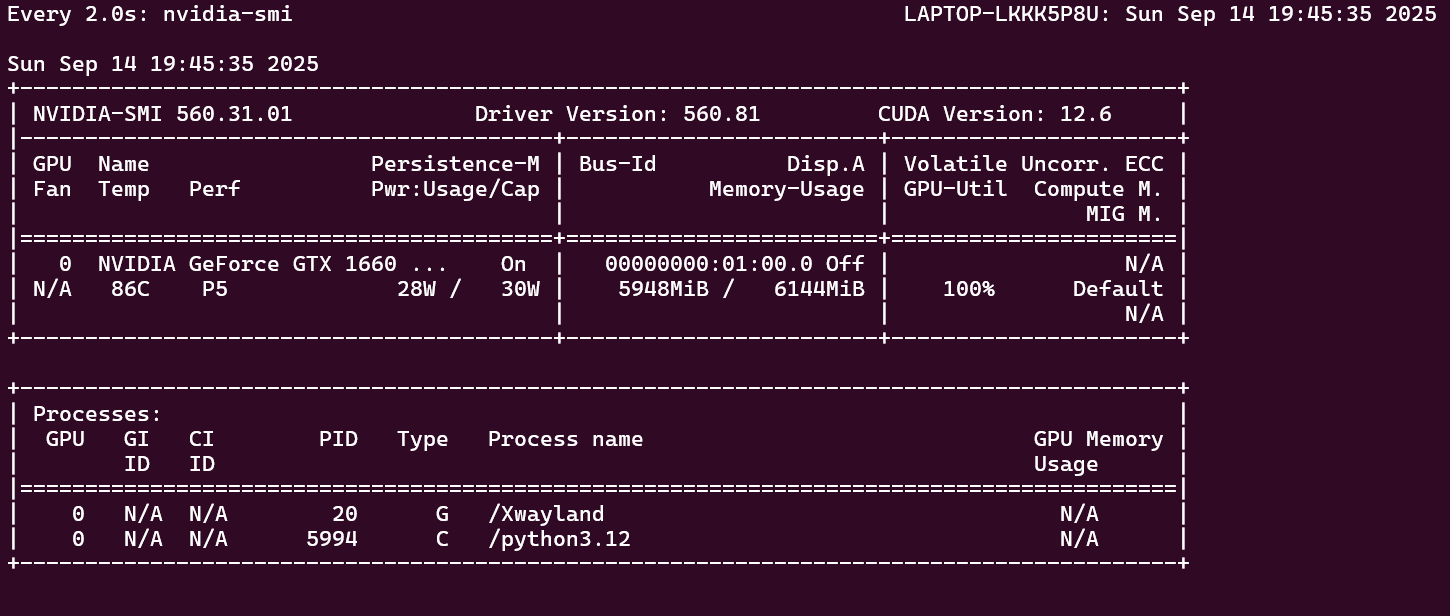
cd /home/francis/tomoro/ConvFinQA/code/finqanet\_retriever

python Main.py \

  --train\_file ../../data/train\_turn.json \

  --valid\_file ../../data/dev\_turn.json \

  --output\_dir ./output



nvidia-smi confirms the retriever training is using your **GTX 1660 (6 GB)** GPU at 100% load. Perfect 👍

**🔎 What is saved in the .pt file?**

The file you see (model.pt) is a **PyTorch checkpoint** created by:

torch.save(model.state\_dict(), os.path.join(save\_dir, "model.pt"))

That means:

* It does **not** include the full model object.
* It includes only the **weights (parameters)** of your retriever model (Bert\_model).
* It’s a mapping: { "layer\_name": tensor, ... }.

So the .pt file holds:

* The RoBERTa encoder weights (fine-tuned on ConvFinQA retrieval task).
* The retriever’s classifier head weights (layers predicting supporting facts).
* No optimizer state (so you can’t resume training exactly, unless you modify code).

**🔧 How it’s used later**

In Test.py, the code will:

model = Bert\_model(...)

model.load\_state\_dict(torch.load(model\_path))

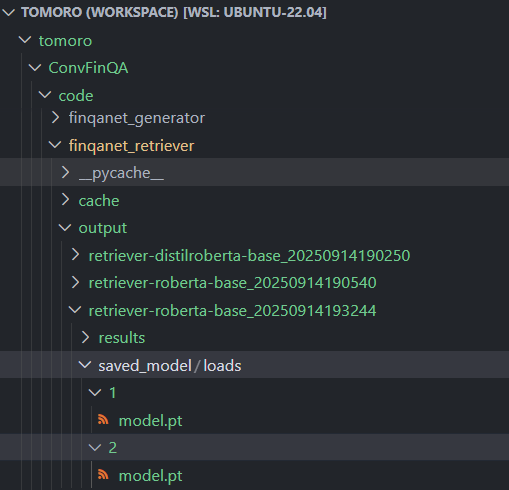
model.eval()

* model\_path points to your model.pt.
* This re-creates the retriever with the learned weights.
* Then it runs inference on your dev\_turn.json or test\_turn.json file.
* Outputs: retriever\_outputs.json → the predicted relevant sentences/table rows.

**✅ Summary**

* The .pt file = trained retriever **weights only** (no full model/optimizer).
* You use it in Test.py to generate retriever predictions.
* Then Convert.py transforms those predictions into generator inputs.

## Models



**🔜 Next steps in the pipeline**

# 1. Evaluate the retriever

Run the test script with your saved model:

python Test.py \

  --model\_path output/retriever-roberta-base\_20250914193244/saved\_model/loads/2/model.pt \

  --test\_file ../../data/dev\_turn.json \

  --save\_path retriever\_outputs.json

# Replace <timestamp> with the actual folder name you saw.

# This will run the retriever on the dev set and write predictions (retriever\_outputs.json).

That command is running your **trained retriever** in inference/evaluation mode.  
Let’s break it down line by line:

**1. Load the trained model**

--model\_path output/retriever-roberta-base\_20250914193244/saved\_model/loads/2/model.pt

* Loads the retriever model weights you trained (model.pt).
* Initializes the retriever architecture (Bert\_model) and fills in all the learned parameters.
* Switches the model into eval() mode (no training, just inference).

**2. Read the test/dev dataset**

--test\_file ../../data/dev\_turn.json

* Opens the **dev set** of ConvFinQA.
* Each example = a conversation turn (question + history + financial report).
* The dev file also includes **gold supporting facts** so evaluation can measure accuracy.

**3. Run retrieval**

* For each example, the model:
  1. Encodes the question and candidate report sentences/rows with RoBERTa.
  2. Scores which sentences/rows are most relevant.
  3. Selects the top-k (based on conf.topn, usually 5).

So the retriever is trying to answer: *“Which parts of the report are relevant for this question?”*

**4. Save predictions**

--save\_path retriever\_outputs.json

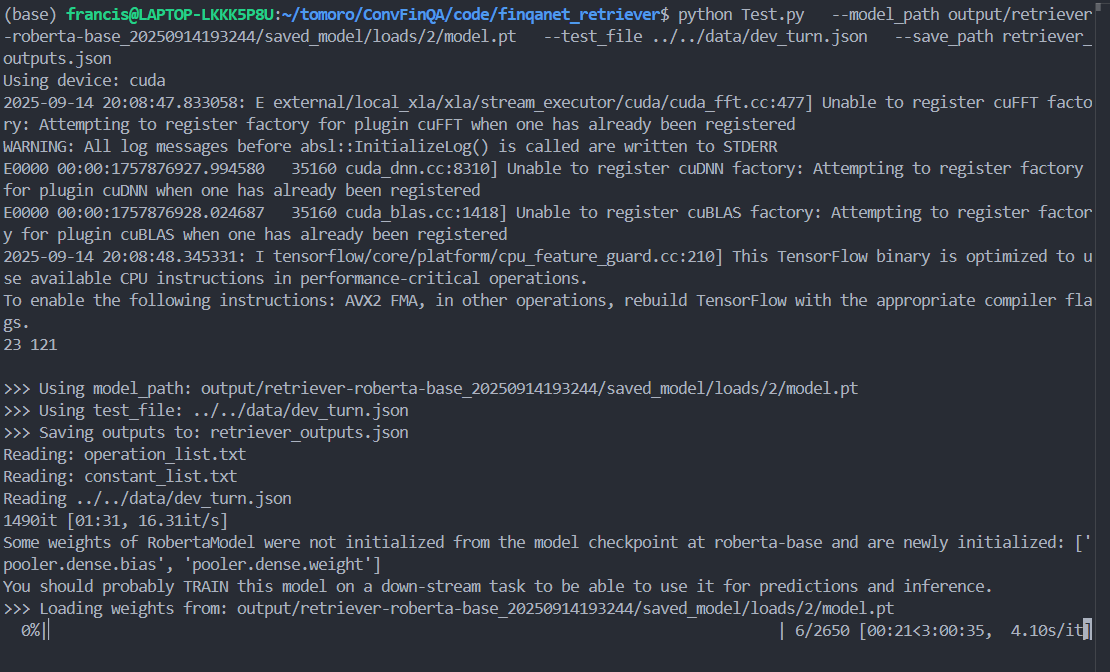
* Writes predictions to retriever\_outputs.json.
* Each entry looks something like:
* {
* "id": "turn\_001",
* "predicted\_supports": [
* "page\_85\_sentence\_3",
* "page\_86\_sentence\_1",
* "table\_row\_5"
* ]
* }

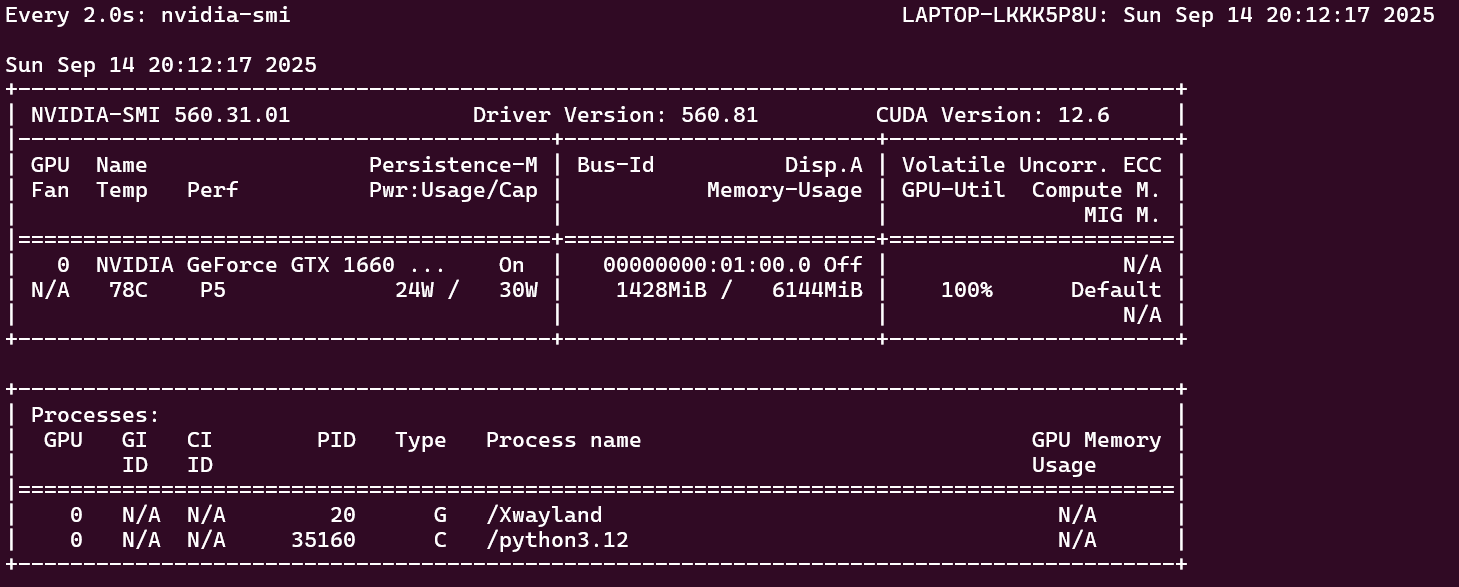
**5. Evaluate performance**

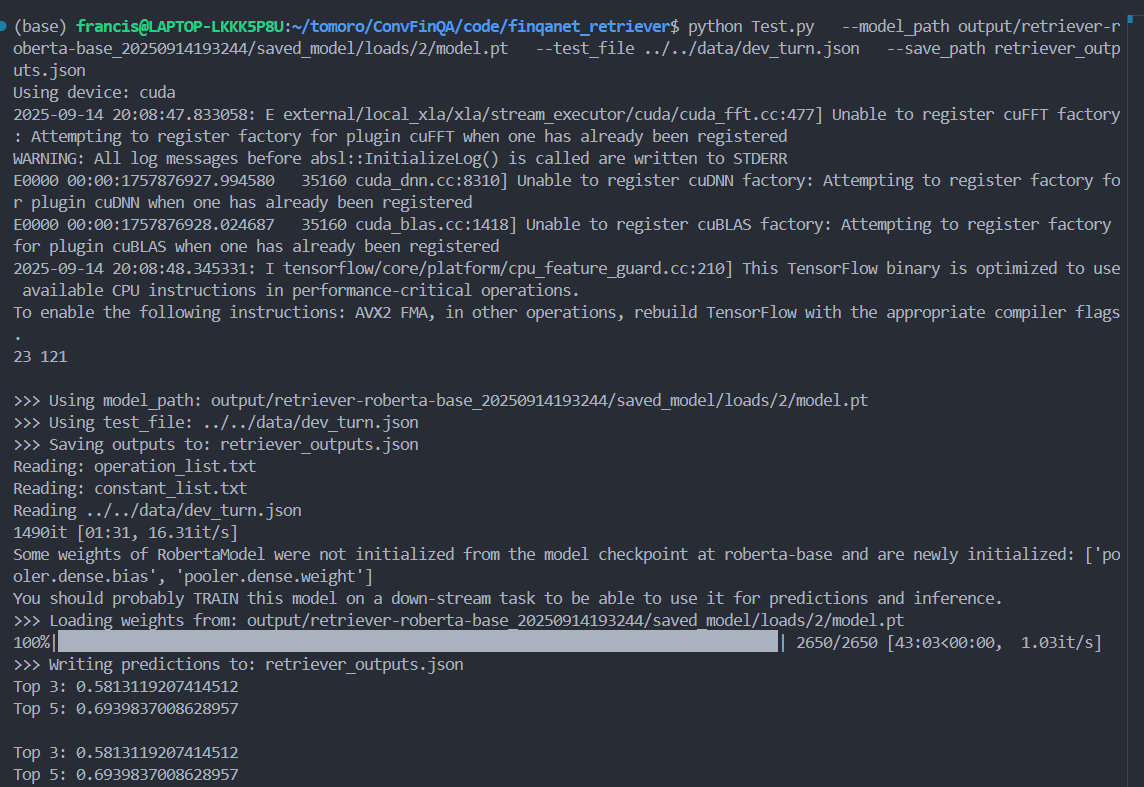
* Since you used **dev set**, Test.py will compare predictions with the gold supporting facts in dev\_turn.json.
* Metrics like **recall@k** (how often the gold evidence is in top-k) will be printed and logged.
* If you had used test\_turn.json, there would be no gold labels → only predictions would be saved.

✅ **Summary:**  
This command takes your trained retriever checkpoint, runs it on the dev set, and outputs both:

* Quantitative metrics (retriever accuracy).
* A JSON file (retriever\_outputs.json) containing the predicted supporting facts, which will be the input for the **generator** stage.







# 2. Convert retriever outputs for generator

Now feed the retriever predictions into the generator stage:

cd /home/francis/tomoro/ConvFinQA/code/finqanet\_generator

python Convert.py \

  --retriever\_file ../finqanet\_retriever/retriever\_outputs.json \

  --save\_path ./dev\_retrieve.json \

  --split dev

[convert\_test] Wrote 1490 examples to ./dev\_retrieve.json

* dev\_retrieve.json → 1490 examples (one per conversation turn in dev\_turn.json)  
  This file contains:
  + the question
  + the gold answer
  + annotated reasoning program
  + the **retrieved evidence**, already reduced to topn=3 snippets (table rows + text)

This is exactly what the **generator’s Main.py** expects for training or evaluation.

# 3. Train the generator

export HF\_HOME=$HOME/.cache/huggingface

export TRANSFORMERS\_CACHE=$HOME/.cache/huggingface/transformers

# make sure you use the HF cache you configured

echo $HF\_HOME

# optionally, run with more connections

pip install hf\_transfer

export HF\_HUB\_ENABLE\_HF\_TRANSFER=1

python Main.py \

  --train\_file ./dev\_retrieve.json \

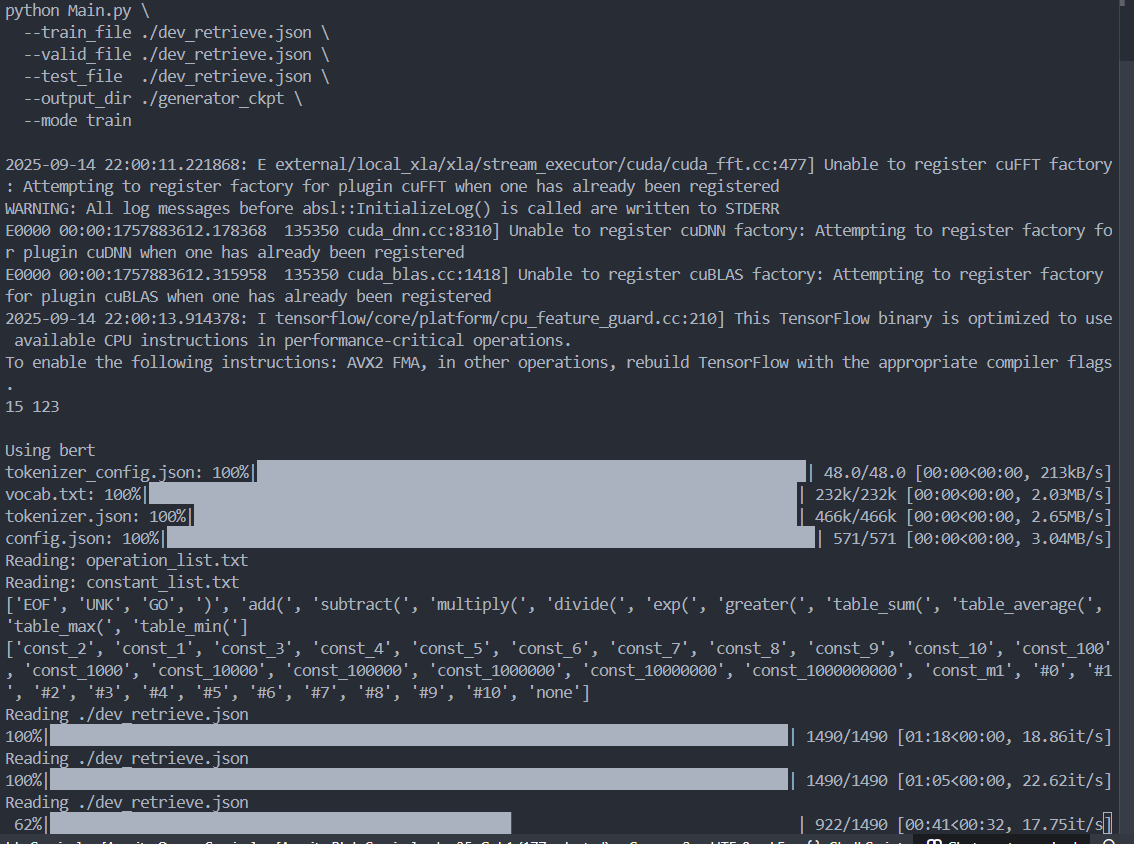
  --valid\_file ./dev\_retrieve.json \

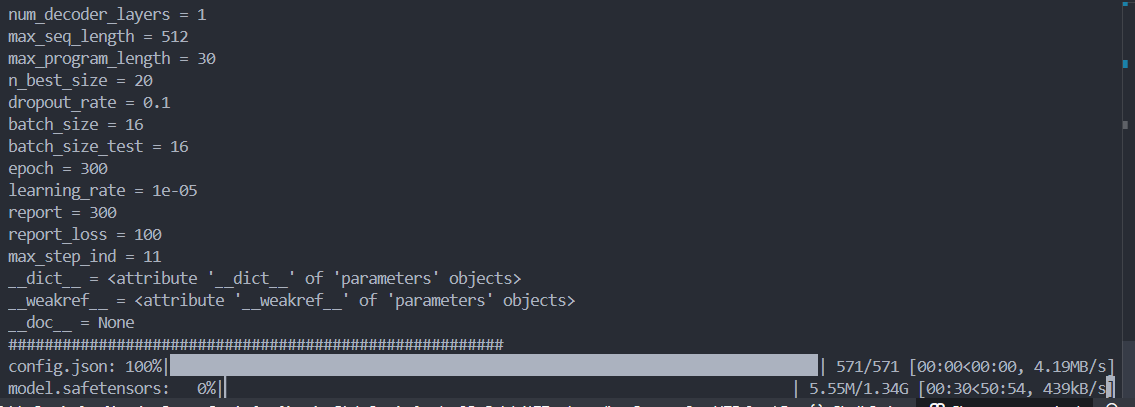
  --test\_file  ./dev\_retrieve.json \

  --output\_dir ./generator\_ckpt \

  --mode train

* The generator learns to map input → gold programs.





# 4. Evaluate generator

export FINQA\_GEN\_CKPT=./generator\_ckpt/generator-bert-base-try\_20250914225507/saved\_model/loads/21/model.pt

cd /home/francis/tomoro/ConvFinQA/code/finqanet\_generator

python Convert.py \

  --retriever\_file ../finqanet\_retriever/retriever\_outputs.json \

  --save\_path ./test\_retrieve.json \

  --split test

export FINQA\_GEN\_CKPT=./generator\_ckpt/generator-bert-base-try\_20250914225507/saved\_model/loads/21/model.pt

python Main.py \

  --test\_file ./test\_retrieve.json \

  --output\_dir ./generator\_ckpt \

  --mode test

