

2025.05.21.(수)

NS3 Code Generator

: LLM 기반의 네트워크 시뮬레이션 코드 자동 생성

202101109 박수화
202102722 최희원

목차

1 Data Cleaning, LLaVa

2 Finetuning

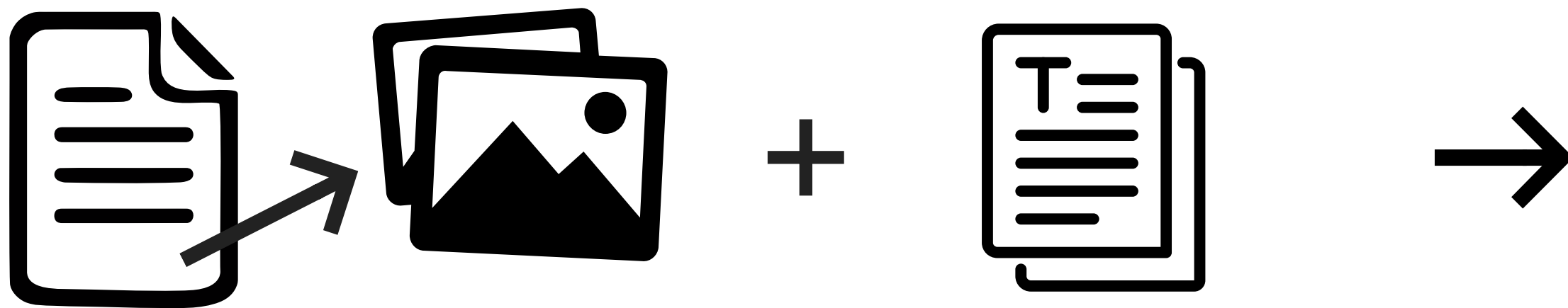
3 DB 구축

4 이후 진행

Data Cleaning

Image → text

- 이전: 이미지 제거 후 텍스트만 사용
- 진행: 이미지를 설명하는 텍스트를 생성해 이미지 정보까지 포함
- 문서(ns-3-manual.pdf)에서 이미지 추출 + 설명 생성해서 저장



```
img_001.png
img_001.txt
img_002.png
img_002.txt
img_003.png
```

LLaVa - 코드

LLaVa.py

```
import fitz
from PIL import Image
import io
import torch
from transformers import AutoProcessor, LlavaForConditionalGeneration
import os
import json

pdf_path = '/home/gpuadmin/kja/ns-3-manual.pdf'
output_dir = './output_images'

os.makedirs(output_dir, exist_ok=True)

# PDF에서 이미지 추출
print("PDF에서 이미지 추출 중...")
doc = fitz.open(pdf_path)
images = []
for page_index in range(len(doc)):
    page = doc[page_index]
    image_list = page.get_images(full=True)
    for img_index, img in enumerate(image_list):
        xref = img[0]
        base_image = doc.extract_image(xref)
        image_bytes = base_image["image"]
        image = Image.open(io.BytesIO(image_bytes))
        images.append(image)

print(f"총 {len(images)}개의 이미지를 추출했습니다.")
```

사용 모델

 [llava-hf/llava-1.5-7b-hf](#) 

- LLaVa (Large Language and Vision Assistant)
 - LLaMA 언어 모델과 CLIP 비전 인코더를 결합한 모델
- 이미지와 텍스트를 입력 받아 작업 가능

LLaVa - 코드

LLaVa.py

```
model_id = "llava-hf/llava-1.5-7b-hf"
model = LlavaForConditionalGeneration.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    low_cpu_mem_usage=True,
).to(0)

processor = AutoProcessor.from_pretrained(model_id)

# 이미지 처리 (배치 크기 5)
BATCH_SIZE = 5
for i in range(0, len(images), BATCH_SIZE):
    batch_images = images[i:i+BATCH_SIZE]
    batch_prompts = []
    batch_conversations = []

    # 모델 사용법 예시를 참고해서 작성하면 된다. (hugging face 에서 모델을 검색하고 모델 카드에 가면 예시가 있다, 각 모델마다 조금씩 상이하여 이는 직접 구현해주는
    for batch_idx, image_in_batch in enumerate(batch_images):
        global_idx = i + batch_idx
        print(f"배치 내 이미지 처리 중: {global_idx+1} / {len(images)}")
        conversation = [
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": "Please describe this image in as much detail as possible. The source of this image is NS3 documentation."},
                    {"type": "image"}
                ]
            }
        ]
        batch_conversations.append(conversation)
    prompt_text_only = processor.apply_chat_template(conversation, add_generation_prompt=True)
    batch_prompts.append(prompt_text_only)
```

LLaVa - 코드

LLaVa.py

```
inputs = processor(text=batch_prompts, images=batch_images, return_tensors="pt", padding=True).to(0, torch.float16)

with torch.no_grad():
    outputs = model.generate(
        **inputs,
        max_new_tokens=512,
        do_sample=False
    )

responses = processor.batch_decode(outputs, skip_special_tokens=True)

for batch_idx, image_in_batch in enumerate(batch_images):
    global_idx = i + batch_idx
    response_text = responses[batch_idx]

    img_filename_base = f"img_{global_idx+1:03d}"
    img_path = os.path.join(output_dir, f"{img_filename_base}.png")
    txt_path = os.path.join(output_dir, f"{img_filename_base}.txt")

    image_in_batch.save(img_path)

    # "ASSISTANT: " 이후의 텍스트만 추출
    assistant_marker = "ASSISTANT:"
    final_text_to_save = response_text.split(assistant_marker, 1)[-1].strip()

    with open(txt_path, 'w', encoding='utf-8') as f:
        f.write(final_text_to_save)
```


Finetuning

이미지와 캡션으로 구성된 GBC 데이터셋, PEFT(-LoRA)

Datasets: [graph-based-captions/](#) **GBC10M** like 33 Follow GBC dataset 10

Tasks: [Image-to-Text](#) Modalities: [Image](#) [Text](#) Formats: [parquet](#) Languages: [English](#) Size: [10M - 100M](#) ArXiv: [arxiv:2407](#)




Libraries: [Datasets](#) [Dask](#) [Croissant](#) +1 License: [cc-by-nc-4.0](#)

[Dataset card](#) [Data Studio](#) [Files and versions](#) [Community](#) 2

Dataset Viewer (First 5GB) Auto-converted to Parquet </> API Embed Data Studio

Split (2)
train · ~17.1M rows (showing the first 710k)

Search this dataset

img_url	img_path	original_caption	short_caption	detail_caption
string · lengths 20 1.84k	null	string · lengths 1 1.44k	string · lengths 1 5.74k	string · lengths 28 6.01k
 https://thumbs.dreamstim.al-112800090.jpg	null	Manager in store with TVs, computers, laptops...	A man in a black suit with a red tie stands...	The image portrays a man standing confidently...
 https://teepublicity.com.y-hair-shirt.jpg	null	Hippie girl dogs On a dark desert highway coo...	A t-shirt with a van design featuring dogs o...	The image showcases a white t-shirt with a...
 https://www.video-games-.lied-Assault.jpg	null	In-game screen of the game <PERSON> - Allied...	A strategic board game scenario with hexagonal...	The image displays a screenshot from a video...



- 참고 자료

: 상품과 설명을 넣어 상세 설명을 만들도록 파인튜닝

참고 링크: <https://blog.futuresmart.ai/fine-tune-llama-32-vision-language-model-on-custom-datasets>

LLaVa_LoRA - 코드

```
import torch
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import Dataset
from transformers import LlavaForConditionalGeneration, LlavaProcessor, TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from PIL import Image
from datasets import load_dataset
import requests
from io import BytesIO

# 모델과 Processor 로드
model_name = "llava-hf/llava-1.5-7b-hf"
model = LlavaForConditionalGeneration.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype=torch.float16,
    load_in_8bit=True,
)
model = prepare_model_for_kbit_training(model)

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
)
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()

processor = LlavaProcessor.from_pretrained(model_name)

# GBC10M Dataset 불러오기
ds = load_dataset("graph-based-captions/GBC10M", split="train[:500]")
```

```
# 커스텀 데이터셋 클래스 정의
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, ds, processor):
        self.ds = ds
        self.processor = processor

    def __len__(self):
        return len(self.ds)

    def __getitem__(self, idx):
        example = self.ds[idx]
        img_url = example['img_url']
        caption = example['detail_caption']

        # 이미지 로드
        try:
            image = Image.open(requests.get(img_url, stream=True, timeout=5).raw).convert("RGB")
        except Exception as e:
            print(f"Image load failed: {e}")
            image = Image.new("RGB", (336,336), (255,255,255)) # fallback

        prompt = "<image>\n" + caption
        processed = self.processor(
            text=prompt,
            images=image,
            return_tensors="pt",
            padding="max_length",
            max_length=512,
        )

        input_ids = processed["input_ids"].squeeze(0)
        attention_mask = processed["attention_mask"].squeeze(0)
        pixel_values = processed["pixel_values"].squeeze(0)

        labels = input_ids.clone()
        labels[labels == self.processor.tokenizer.pad_token_id] = -100
```

LLaVa_LoRA - 코드

```
labels = input_ids.clone()
labels[labels == self.processor.tokenizer.pad_token_id] = -100

return {
    "input_ids": input_ids,
    "attention_mask": attention_mask,
    "pixel_values": pixel_values,
    "labels": labels,
}

# Collator 정의
class CustomDataCollator:
    def __init__(self, tokenizer):
        self.tokenizer = tokenizer

    def __call__(self, batch):
        input_ids = [item["input_ids"] for item in batch]
        attention_mask = [item["attention_mask"] for item in batch]
        labels = [item["labels"] for item in batch]
        pixel_values = [item["pixel_values"] for item in batch]

        input_ids_padded = pad_sequence(input_ids, batch_first=True, padding_value=self.tokenizer.pad_token_id)
        attention_mask_padded = pad_sequence(attention_mask, batch_first=True, padding_value=0)
        labels_padded = pad_sequence(labels, batch_first=True, padding_value=-100)
        pixel_values_tensor = torch.stack(pixel_values)

        return {
            "input_ids": input_ids_padded,
            "attention_mask": attention_mask_padded,
            "labels": labels_padded,
            "pixel_values": pixel_values_tensor,
        }

# Dataset 인스턴스화
dataset = CustomDataset(ds, processor)
```

```
# 훈련 설정
training_args = TrainingArguments(
    output_dir="./llava-lora-finetuned",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=3,
    learning_rate=2e-5,
    save_total_limit=2,
    fp16=True,
    logging_dir="./logs",
    logging_steps=10,
    save_steps=500,
)

# Trainer 정의 및 훈련
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset,
    data_collator=CustomDataCollator(processor.tokenizer),
)

trainer.train()

# 모델 저장
model.save_pretrained("./llava-lora-finetuned/adapters")
```

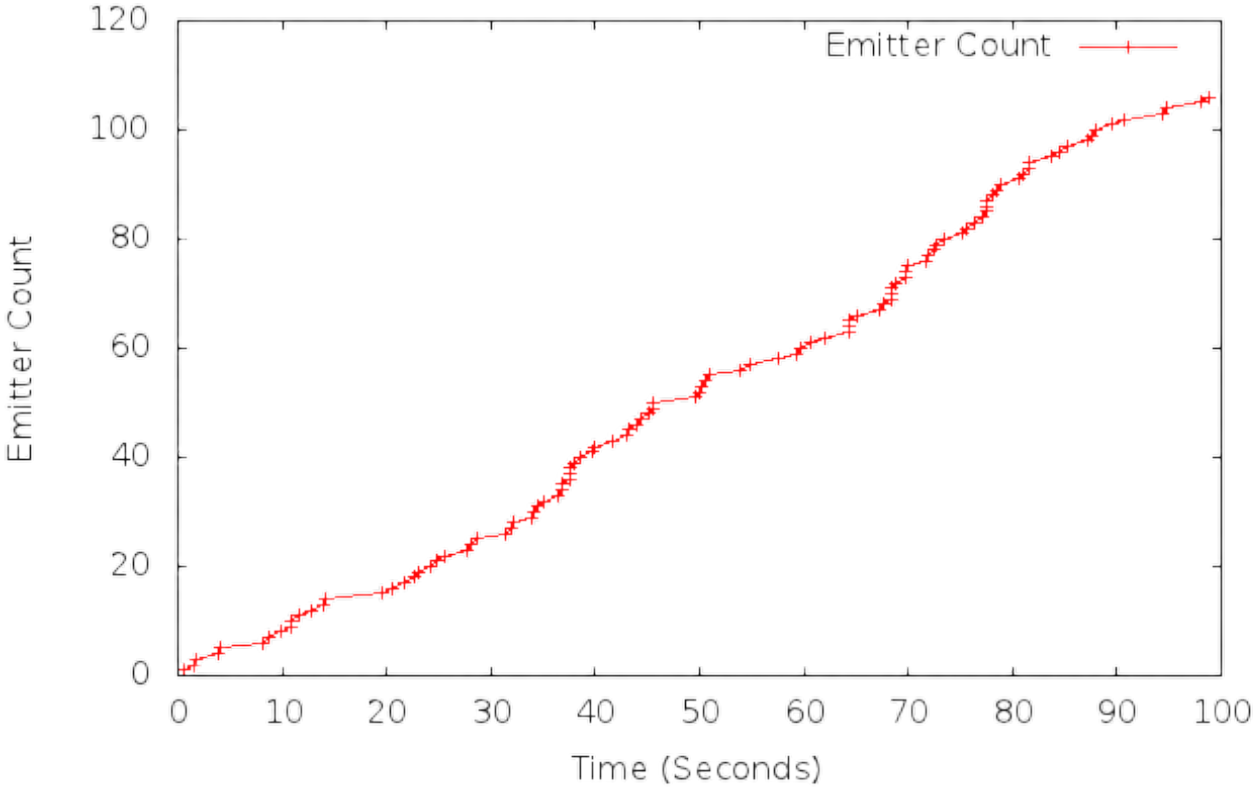
Finetuning - 결과

iimg_005.png

파인튜닝 전

이미지에는 방출자 수 대비 시간을 나타내는 그래프가 표시됩니다. 그래프는 빨간색이며 기울기가 커서 시간이 지남에 따라 방출자 수가 급격히 증가함을 나타냅니다. X축은 시간을 나타내고 Y축은 방출자 수를 나타냅니다. 그래프에는 시간을 초 단위로 표시하고 방출자 수를 로그 스케일로 표시합니다. 데이터는 명확하고 간결한 방식으로 표시되므로 시간에 따른 방출자 수의 추세를 쉽게 이해할 수 있습니다.

Emitter Counts vs. Time
Probe Path: /Names/Probe/Output



파인튜닝 후

이미지는 방출자 수 대 시간을 그래픽으로 표시합니다. x축은 0에서 1000밀리초 사이의 값으로 시간을 나타냅니다. y축은 방출자 수를 나타내며 0에서 1000 사이의 값을 갖습니다. 그래프는 빨간색으로 표시되어 방출자 수와 시간 사이에 음의 관계가 있음을 나타냅니다. 그래프는 선형 모양으로 시간에 따라 방출자 수가 일정하게 증가함을 나타냅니다. 그래프의 스케일은 로그형이며, 각 시간 단위마다 값이 10배씩 증가합니다. 시간 단위는 0에서 1000 사이의 값으로 밀리초 단위입니다. 방출자 수 단위도 0에서 1000 사이의 값으로 밀리초 단위입니다. 그래프에는 시간 단위와 방출자 수가 표시되어 있어 두 변수 간의 관계를 쉽게 이해할 수 있습니다.

Finetuning - 결과

iimg_007.png

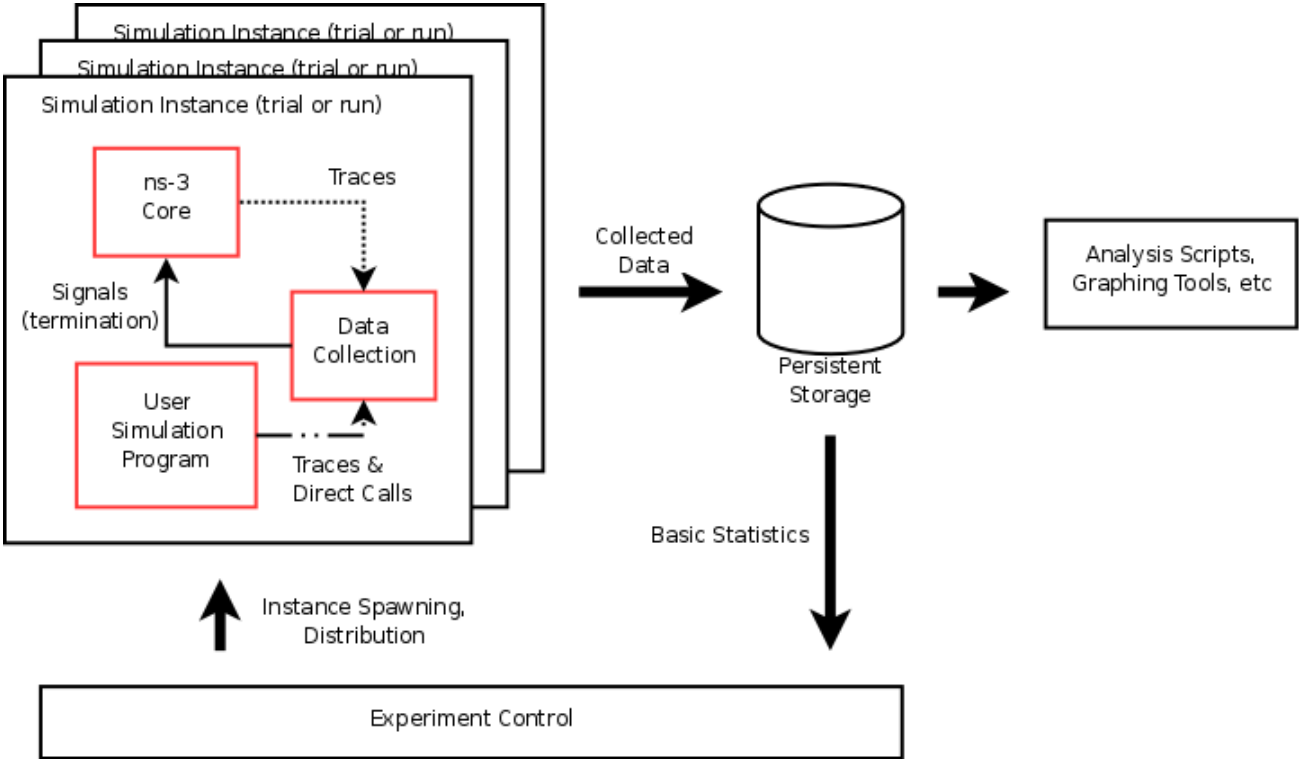
파인튜닝 전

이미지는 데이터 수집 시스템의 흐름도를 보여줍니다. 흐름도는 여러 상자와 화살표로 구성되어 데이터 수집, 저장 및 검색 프로세스를 보여줍니다. 상자는 데이터 수집 프로세스의 여러 단계를 나타내고 화살표는 이러한 단계 사이의 흐름을 나타냅니다.

흐름도는 다양한 소스에서 데이터를 수집하는 데이터 수집 프로세스로 시작됩니다. 수집된 데이터는 데이터베이스에 저장되며, 데이터베이스는 흐름도 중앙에 큰 상자로 표시됩니다. 데이터베이스는 화살표를 통해 데이터 수집 프로세스에 연결되어 두 단계 사이의 데이터 흐름을 나타냅니다.

이 흐름도에는 저장된 데이터를 다양한 용도로 액세스하고 사용할 수 있는 데이터 검색 섹션도 포함되어 있습니다. 이 섹션은 흐름도 오른쪽에 있는 작은 상자로 표시되며 화살표가 데이터베이스 및 데이터 수집 단계에 연결되어 있습니다.

전반적으로, 흐름도는 데이터 수집 시스템의 명확한 시각적 표현을 제공하며, 과정에 관련된 다양한 단계들을 강조합니다.



파인튜닝 후

이미지는 네트워크 장치와 사용자 간의 상호 작용을 설명하는 다이어그램을 보여줍니다. 장치는 파란색 배경을 가진 큰 원으로 표시되며, 사용자는 빨간색 배경을 가진 작은 도형으로 표시됩니다. 다이어그램은 장치가 사용자에게 데이터를 전송하면 사용자는 데이터를 수신하고 요청을 다시 장치로 보내 응답하는 것을 보여줍니다. 상호 작용은 장치와 사용자를 연결하는 화살표로 표시되며, 레이블은 데이터 흐름의 방향을 나타냅니다. 다이어그램에는 또한 "지속 저장"이라는 텍스트가 있는 작은 직사각형이 포함되어 있어 나중에 사용하기 위해 장치에 데이터를 저장할 수 있음을 시사합니다. 다이어그램의 전체 레이아웃은 장치와 사용자 간의 상호 작용을 명확하고 체계적으로 표현할 수 있음을 시사합니다.

DB - 크롤링

ns3 커뮤니티 사이트(<https://groups.google.com/g/ns-3-users>)에서

Tommaso Pecorella(ns3 만든 사람) 의 답변이 달린 질문 - 답변 쌍 크롤링해서 json으로 추출

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import tempfile
import shutil
import time
import json
import re

# 1. Chrome 옵션 설정
options = webdriver.ChromeOptions()
options.add_argument("--disable-gpu")
options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
options.add_argument("--headless=new")

# 2. 임시 user-data-dir 생성 (세션 격리)
temp_user_data_dir = tempfile.mkdtemp(prefix="selenium-profile-")
options.add_argument(f"--user-data-dir={temp_user_data_dir}")

# 3. ChromeDriver 실행
service = Service("/usr/bin/chromedriver")
driver = webdriver.Chrome(service=service, options=options)

# 4. 본문 텍스트만 추출 (이미지·링크 제거)
def extract_clean_text(element):
    for tag in element.find_elements(By.TAG_NAME, "img") + element.find_elements(By.TAG_NAME, "a"):
        driver.execute_script("arguments[0].remove();", tag)
    return element.text.strip()
```

DB - 크롤링

5. 본문 이미지 여부 검사 (오직 /groups/ 경로의 이미지만)

```
def has_visible_content_images(element):
    imgs = element.find_elements(By.TAG_NAME, "img")
    for img in imgs:
        if not img.is_displayed():
            continue
        src = img.get_attribute("src") or ""
        if "groups/" in src:
            return True
    return False
```

6. 게시글 링크 수집

```
def collect_thread_links(start_url, max_pages=3):
    links = []
    driver.get(start_url)
    time.sleep(3)
    for _ in range(max_pages):
        for a in driver.find_elements(By.CSS_SELECTOR, "a[href*='/g/ns-3-users/c/']"):
            href = a.get_attribute("href")
            if href and href not in links:
                links.append(href)
        try:
            nxt = driver.find_element(By.CSS_SELECTOR, "div[aria-label='다음 페이지']")
            driver.execute_script("arguments[0].click()", nxt)
            time.sleep(2)
        except:
            break
    return links
```

DB - 크롤링

```
# 7. 질문-답변 파싱
def parse_thread(url):
    driver.get(url)
    try:
        WebDriverWait(driver, 10).until(
            EC.presence_of_all_elements_located((By.CSS_SELECTOR, "section[data-author]"))
        )
        secs = driver.find_elements(By.CSS_SELECTOR, "section[data-author]")
        if len(secs) < 2:
            print("❌ 답변 없음:", url)
            return None

        # 질문 본문만 추출 & clean
        q_sec = secs[0]
        try:
            q_region = q_sec.find_element(By.CSS_SELECTOR, "div[role='region']")
            raw_q = q_region.text
        except:
            raw_q = q_sec.text
        question = re.sub(r"\s+", " ", raw_q).strip()

        # Tommaso 섹션 찾기
        t_sec = next((s for s in secs[1:]
                       if s.get_attribute("data-author") == "Tommaso Pecorella"), None)
        if not t_sec:
            print("❌ Tommaso 답변 없음:", url)
            return None
```

```
# 본문 이미지 포함 여부
if has_visible_content_images(q_sec):
    print("🖼️ 질문 본문에 이미지 포함 → 제외:", url)
    return None
if has_visible_content_images(t_sec):
    print("🖼️ Tommaso 답변 본문에 이미지 포함 → 제외:", url)
    return None

# 답변 본문만 추출 & clean
try:
    a_region = t_sec.find_element(By.CSS_SELECTOR, "div[role='region']")
    raw_a = a_region.text
except:
    raw_a = t_sec.text
answer = re.sub(r"\s+", " ", raw_a).strip()

if not answer:
    print("⚠️ 답변 비어 있음:", url)
    return None

print("✅ Tommaso 답변 수집:", url)
return {
    "url": url,
    "question": question,
    "answer": answer
}

except Exception as e:
    print(f"⚠️ 페이지 로드 실패 ({url}): {e}")
return None
```

DB - 크롤링

```
# 8. 메인 실행 로직
if __name__ == "__main__":
    base_url = "https://groups.google.com/g/ns-3-users"
    threads = collect_thread_links(base_url, max_pages=3)

    results = []
    seen = set()
    for link in threads:
        if link in seen:
            continue
        seen.add(link)

        print(f"\n🔍 크롤링 중: {link}")
        qa = parse_thread(link)
        if qa:
            results.append(qa)

    with open("tommaso_qa.json", "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=2)

    print(f"\n📦 저장 완료: {len(results)}개 수집됨")

    driver.quit()
    shutil.rmtree(temp_user_data_dir, ignore_errors=True)
```


DB - 크롤링

저장 결과 중 일부

```
{
  "url": "https://groups.google.com/g/ns-3-users/c/-XQ1cyau3HY",
  "question": "Hello, I have been working to send machine learning model weights in an ad-hoc network using TCP connections. My code is based in Python and I am not using the ns3-ai module (in hindsight I might try this). I am facing issues sending the model weights across the network and would like help in doing this. My current script has issues sending the model weights due to their structure. The model outputs weights in a list of arrays: weights = (10, _) and indexing into the weights gives arrays of shapes like (32, _) and (16, 16, 32, 64). I have tried such things: * Sending the weights directly - errors: Cannot send list data type (needs to be serialized). Trying to serialize the weights directly leads to corruption. I tried using bytearray() to serialize the data. * Breaking up the weights for transmission (packetization) - scrambled: Given the ad-hoc structure, packets reach an ego node \"randomly,\" and when analyzing the packets at the server, the data is scrambled - wrong order. (I have a grid of 25 nodes and one of them is assigned to be a \"server,\" one an \"ego\" node, and the rest * Compression - errors: I tried using H5Py to compress the weights into a file for transmission. Using flush() and id.get_file_image() get the bytes of the file. This send the data but upon trying to view the contents of the file breaks due to synchronous file opening. If it helps, I can provide my current code. Thank you.",
  "answer": "Hi, I don't use ns-3 tru python, but I guess you have to convert your data structure into an array of bytes. However, it might be even more convenient for you to serialize your data into a json, as decoding will be more robust (eventually you'll compress the json). As a matter of fact, sending data between nodes involves a lot of problems, including byte orders, array sizes, floating point representations, and so on, which are solved either by defining a data representation mechanism, or by serializing the data into a common format - and then why not json, it's compact enough. "
},
```

이후 진행

- Data cleaning
 - 생성한 이미지 설명을 같은 페이지의 다른 텍스트와 합하기
- DB
 - 전체 크롤링해서 구축하기