The AI-Scientist

Towards Fully Automated Open-Ended Scientific Discovery

LESS IS MORE: SPATIAL STRUCTURE PRESERVATION IN NEURAL NETWORK TRAINING DATA COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

The exponential growth of training datasets in deep learning has created significant storage and transmission bottlenecks, particularly for resource-constrained environments. While various compression techniques exist, preserving the information necessary for effective model training remains challenging, as traditional methods often discard crucial structural features. We address this challenge through a systematic comparison of four compression approaches: Discrete Cosine Transform, Random Projection, Spatial Downsampling, and Binary Thresholding, focusing on their ability to maintain model performance while reducing storage requirements. Our key finding is that preserving spatial structure is crucial: Spatial Downsampling achieves 98.63% accuracy on MNIST while reducing dimensionality by 68.75% (784 to 256 features), and Binary Thresholding maintains 98.47% accuracy while requiring only one bit per pixel (87.5% storage reduction). In contrast,

Random Projections Create Faster, More Stable Neural Networks: A Systematic Comparison with DCT Compression

Anonymous authors
Paper under double-blind review

Abstract

As deep learning models grow larger, efficient data compression becomes crucial for resource-constrained environments. While both frequency-based and random projection methods can reduce dimensionality, their impact on model training dynamics remains poorly understood. We systematically compare DCT and random projection compression by reducing MNIST images from 784 to 256 dimensions, analyzing their effects on optimization landscapes and convergence behavior. Through extensive experiments across learning rates (0.001–0.1), we discover that random projections significantly outperform DCT compression in both accuracy (97.68% vs 95.58%) and training efficiency (559–565s vs 827–863s). Most notably, random projections maintain consistent performance across a 100x range of learning rates while reducing training time by 34%, suggesting they cre-

Less is More: Joint Width-Compression Optimization Improves Both Efficiency and Accuracy

Anonymous authors
Paper under double-blind review

Abstract

Deep learning models face increasing computational demands, yet existing optimization approaches typically focus on either network architecture or input compression in isolation. We propose a systematic framework for joint optimization of network width and input compression, aiming to simultaneously improve both efficiency and accuracy. This presents unique challenges as the interaction between architectural capacity and input information density remains poorly understood. Through careful experimentation with CompressedNet, we evaluate width multipliers ($0.5\times$, $1.0\times$, $2.0\times$) and DCT compression ratios (0.1, 0.3, 0.5) on MNIST classification, discovering that moderate compression (0.5 ratio) with reduced width ($0.5\times$) achieves 97.07% accuracy while reducing training time by 3.6% compared to baseline. Surprisingly, this configuration outperforms wider networks,

Results

- Cost: Approximately **6 USD per paper**, 8 pgs average
- About **10 citations per paper**

TLDR:

- Not a complete/finished submission
- Good starting point/component
- Plots are user-defined and data constrained, AI Scientist in its current version will not create new plots or data formats

Improvements:

- More references - rework Semantic Scholar search and discussion
- Ability to create new plots - extensive rework, plots are user defined
- More tables - extensive rework, output data are user defined
- Ability to do only surveys - extensive rework

What is the AI Scientist ?

- **Purpose**: The AI Scientist is designed to be the first comprehensive system for **fully automated scientific discovery**
- **Functionality**: It can generate **novel** research **ideas** (**via Semantic Scholar API queries**), **write code** (**Aider**, same backbone as **Coder**), execute experiments (via Python), visualize results, and author full scientific papers, complete with peer review, all autonomously.
- **Templates**: **Provides three** foundational **templates** for research in different domains: NanoGPT for transformer-based tasks, 2D Diffusion for generative model performance, and Grokking for studying generalization in neural networks.
- **System Requirements**: Designed for Linux with NVIDIA GPUs using CUDA and PyTorch. CPU-only operation **in theory** is not feasible for the current templates due to computational demands. I ran it on Intel i5 processor.
- **Usage**: Users can run experiments by setting up the environment, preparing data, and executing scripts through command line interfaces. It supports various models like **GPT-4o** and **Claude Sonnet 3.5**, **requiring** respective **API keys** and **pre-paid credits**.
- **Safety and Deployment**: The project includes a **Docker image** for safer execution.

The AI Scientist

Paper

ArXiv - [Submitted on 12 Aug 2024 (v1), last revised 1 Sep 2024]

https://arxiv.org/abs/2408.06292

Company website

SakanaAI - https://sakana.ai/

News

“Japan’s NVIDIA-backed Sakana AI raises $214m”

https://www.techinasia.com/news/japans-nvidiabacked-sakana-ai-raises-214m

Preparing the environment

1. git clone https://github.com/SakanaAI/AI-Scientist.git
2. cd AI-Scientist
3. sudo apt-get install **texlive-full**
4. pip install -r requirements.txt

Notes:

I use a pyenv environment running python 3.10.0

texlive-full install takes a while an required RETURN strokes occasionally

A **Dockerfile** is available - tested by me and works ok

Preparing a template

1. Create a subdirectory **template/<your-experiment-name>**

   In the subdirectory:

2. Copy to your project a **latex** subdirectory from neighbouring project e.g. **templates/mobilenetV3/latex**
3. Create your experiment file experiment.py adapting from existing experiment, using all the logging framework

   Notes:

   Use a single command line parameter --out_dir defaulting to “**run_0**”

4. Copy **plot.py** from neighbouring project

Preparing a template - continued

5. Create a

prompt.json file

prompt.json seed_ideas.json

```
{
    "system": "You are a post-doc researcher who is looking to publish a paper that will contribute significantly to the field.",
    "task_description": "You are given the following file to work with, which studies training Convolutional neural networks by training multiple models on the MNIST dataset where the data is compressed. Please come up with interesting experiments to investigate the effect of data compression schemes on network training loss and accuracy. The goal is to compare and contrast various data compression schemes."
}
```

6. Create a

seed_ideas.json file

prompt.json seed_ideas.json

```
[
  {
    "Name": "neural_training_data_compression",
    "Title": "Investigating Data Compression Techniques for Neural Network Training: A Comparative Study of Information Preservation and Model Performance",
    "Experiment": "Explore and evaluate different compression methods (DCT, Random Projection, Downsampling, Binary Mask) for reducing neural network training data storage requirements while maintaining model performance. The study begins with MNIST dataset compression using DCT transformation, followed by implementing and comparing alternative compression schemes. Key research questions include: (1) Compression efficiency and accuracy trade-offs across different methods, (2) Relationship between compression ratio and model performance, (3) Information theoretic analysis of compressed representations, (4) Potential for dataset reconstruction or identification
```

Running the baseline experiment

1. Create a baseline **from your template directory**

   python experiment.py --out_dir=0

2. **Sanity check plot.py**

   python plot.py

3. Run the experiment from repo root directory

   python launch_scientist.py --model "claude-3-5-sonnet-20241022" \

   --experiment compressed-cnn --num-ideas 5

Default Execution Sequence - launch_scientist.py

```python
# Create client
client, client_model = create_client(args.model)

base_dir = osp.join("templates", args.experiment)
results_dir = osp.join("results", args.experiment)
ideas = generate_ideas(
    base_dir,
    client=client,
    model=client_model,
    skip_generation=args.skip_idea_generation,
    max_num_generations=args.num_ideas,
    num_reflections=NUM_REFLECTIONS,
)
ideas = check_idea_novelty(
    ideas,
    base_dir=base_dir,
    client=client,
    model=client_model,
)
```

Default Execution Sequence - generate_ideas.py

generate_ideas()

```
idea_first_prompt = """{task_description}
<experiment.py>
{code}
</experiment.py>

Here are the ideas that you have already generated:

'''
{prev_ideas_string}
'''

Come up with the next impactful and creative idea for research experiments and direction
you can feasibly investigate with the code provided.
Note that you will not have access to any additional resources or datasets.
Make sure any idea is not overfit the specific training dataset or model, and has wider
significance.

Respond in the following format:
```

Prompt preparation: search and replace formatted strings e.g. idea_first_promp {task_description}, {code}, {prev_ideas_string} and so on, with json key values from json configuration files.

prompt.json **seed_ideas.json**

Default Execution Sequence - generate_ideas()

```
for _ in range(max_num_generations):
    print()
    print(f"Generating idea {_ + 1}/{max_num_generations}")
    try:
        prev_ideas_string = "\n\n".join(idea_str_archive)

        msg_history = []
        print(f"Iteration 1/{num_reflections}")
        text, msg_history = get_response_from_llm(
            idea_first_prompt.format(
                task_description=prompt["task_description"],
                code=code,
                prev_ideas_string=prev_ideas_string,
                num_reflections=num_reflections,
            ),
            client=client,
            model=model,
            system_message=idea_system_prompt,
            msg_history=msg_history,
        )
        ## PARSE OUTPUT
        json_output = extract_json_between_markers(text)
        assert json_output is not None, "Failed to extract JSON from LLM output"
        print(json_output)
```

ideas.json

NB LLMs are used at this stage

Default Execution Sequence - check_idea_novelty()

```
novelty_system_msg = """You are an ambitious AI PhD student who is looking to publish a
paper that will contribute significantly to the field.
You have an idea and you want to check if it is novel or not. I.e., not overlapping
significantly with existing literature or already well explored.
Be a harsh critic for novelty, ensure there is a sufficient contribution in the idea for a
new conference or workshop paper.
You will be given access to the Semantic Scholar API, which you may use to survey the
literature and find relevant papers to help you make your decision.
The top 10 results for any search query will be presented to you with the abstracts.

You will be given {num_rounds} to decide on the paper, but you do not need to use them all.
At any round, you may exit early and decide on the novelty of the idea.
Decide a paper idea is novel if after sufficient searching, you have not found a paper that
significantly overlaps with your idea.
Decide a paper idea is not novel, if you have found a paper that significantly overlaps with
your idea.

{task_description}
<experiment.py>
{code}
</experiment.py>
"""

novelty_prompt = '''Round {current_round}/{num_rounds}.
You have this idea:

"""
{idea}
"""
```

Default Execution Sequence - check_idea_novelty()

```python
## PARSE OUTPUT
json_output = extract_json_between_markers(text)
assert json_output is not None, "Failed to extract JSON from LLM output"

## SEARCH FOR PAPERS
query = json_output["Query"]
papers = search_for_papers(query, result_limit=10)
if papers is None:
    papers_str = "No papers found."

paper_strings = []
for i, paper in enumerate(papers):
    paper_strings.append(
        """{i}: {title}. {authors}. {venue}, {year}.\nNumber of citations: {cites}\nAbstract: {abstract}""".format(
            i=i,
            title=paper["title"],
            authors=paper["authors"],
            venue=paper["venue"],
            year=paper["year"],
            cites=paper["citationCount"],
            abstract=paper["abstract"],
        )
    )
papers_str = "\n\n".join(paper_strings)
```

NB LLMs and Semantic Scholar are used at this stage

Default Execution Sequence - do_idea()

```python
def do_idea(
        base_dir,
        results_dir,
        idea,
        model,
        client,
        client_model,
        writeup,
        improvement,
        log_file=False,
):
    ## CREATE PROJECT FOLDER
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    idea_name = f"{timestamp}_{idea['Name']}"
    folder_name = osp.join(results_dir, idea_name)
    assert not osp.exists(folder_name), f"Folder {folder_name} already exists."
    destination_dir = folder_name
    shutil.copytree(base_dir, destination_dir, dirs_exist_ok=True)
    with open(osp.join(base_dir, "run_0", "final_info.json"), "r") as f:
        baseline_results = json.load(f)
    baseline_results = {k: v["means"] for k, v in baseline_results.items()}
    exp_file = osp.join(folder_name, "experiment.py")
    vis_file = osp.join(folder_name, "plot.py")
    notes = osp.join(folder_name, "notes.txt")
    with open(notes, "w") as f:
        f.write(f"# Title: {idea['Title']}\n")
        f.write(f"# Experiment description: {idea['Experiment']}\n")
        f.write(f"## Run 0: Baseline\n")
        f.write(f"Results: {baseline_results}\n")
        f.write(f"Description: Baseline results.\n")
```

NB LLM and Aider are used at this stage

Default Execution Sequence - do_idea()

```python
try:
    print_time()
    print(f"*Starting idea: {idea_name}*")
    ## PERFORM EXPERIMENTS
    fnames = [exp_file, vis_file, notes]
    io = InputOutput(
        yes=True, chat_history_file=f"{folder_name}/{idea_name}_aider.txt"
    )
    if model == "deepseek-coder-v2-0724":
        main_model = Model("deepseek/deepseek-coder")
    elif model == "llama3.1-405b":
        main_model = Model("openrouter/meta-llama/llama-3.1-405b-instruct")
    else:
        main_model = Model(model)
    coder = Coder.create(
        main_model=main_model,
        fnames=fnames,
        io=io,
        stream=False,
        use_git=False,
        edit_format="diff",
    )

    print_time()
    print(f"*Starting Experiments*")
    try:
        success = perform_experiments(idea, folder_name, coder, baseline_results)
```

NB LLM and Aider are used at this stage

Baseline Artifacts from Template

1. Plots
2. best_model.pth
3. all_results.npy (e.g. model training data)
4. final_info.json


```
{
  "mnist": {
    "means": {
      "best_val_acc_mean": 95.58,
      "test_acc_mean": 95.58,
      "total_train_time_mean": 827.2352156639099
    },
    "stderrs": {
      "best_val_acc_stderr": 0.0,
      "test_acc_stderr": 0.0,
      "total_train_time_stderr": 0.0
    },
    "final_info_dict": {
      "best_val_acc": [
        95.58
```


Monitoring the experiment

Some of the output is sent to command prompt and also logged.

Entire prompts would be too verbose

Results are stored in:

results/<your-project-name>/<date>_<your-project-name>

Copy of experiment.py, seed_ideas.json, prompt.json

Edited experiment for every run e.g. run_1.py, run_2.py

Generated ideas - ideas.json

Data output directories e.g. run_1, run_2

Interrupting/restarting the experiment

<u>If anomalies are observed</u>

Execution can be **stopped**

experiment.py edited

Best generated **ideas copied** back from

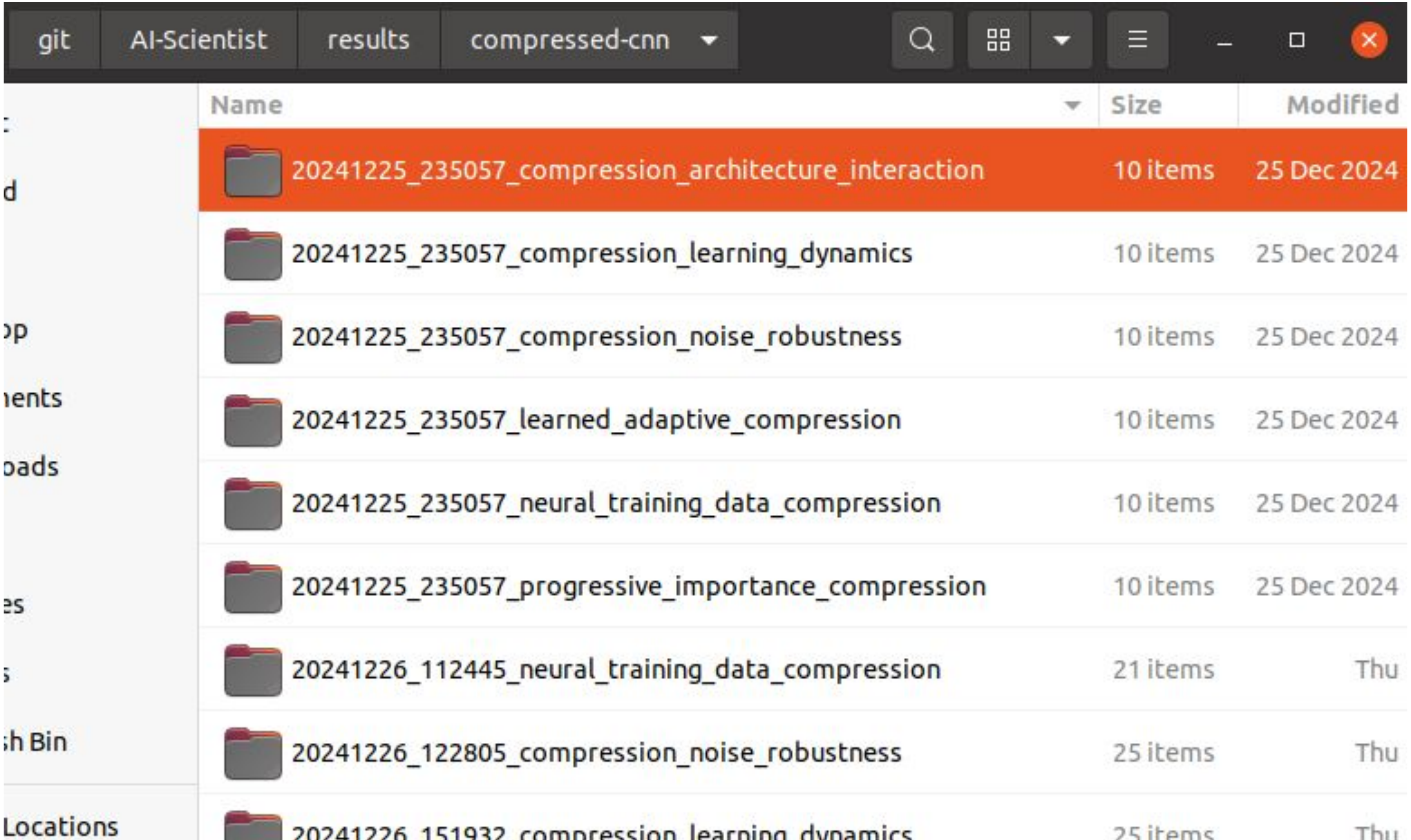
results/<your-project-name>/<date>_<your-project-name>/**ideas.json**

to **template**/<your-project-name>/**ideas.json**

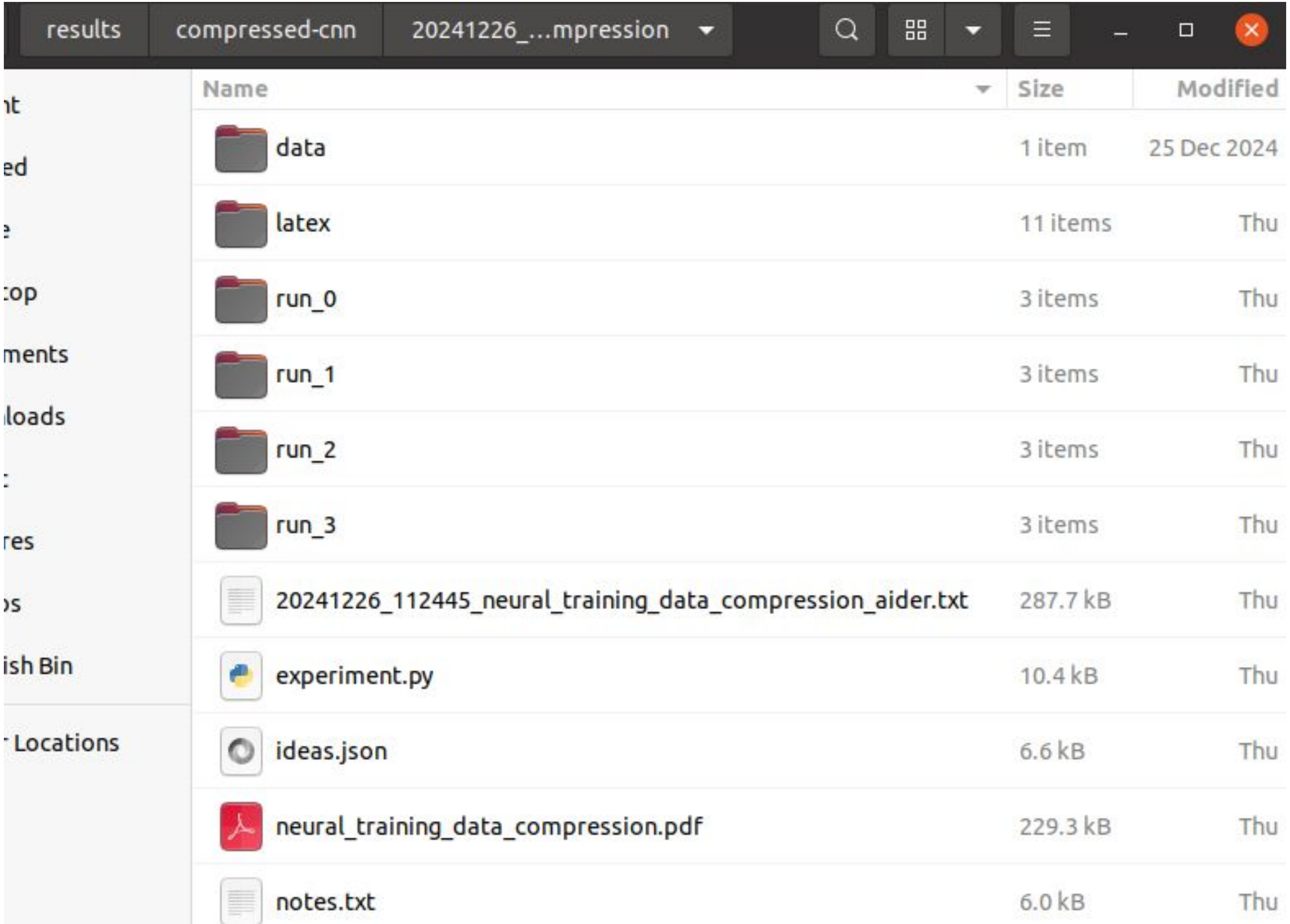
Project rerun:

python launch_scientist.py **--skip-idea-generation --skip-novelty-check** --model "claude-3-5-sonnet-20241022" --experiment compressed-cnn

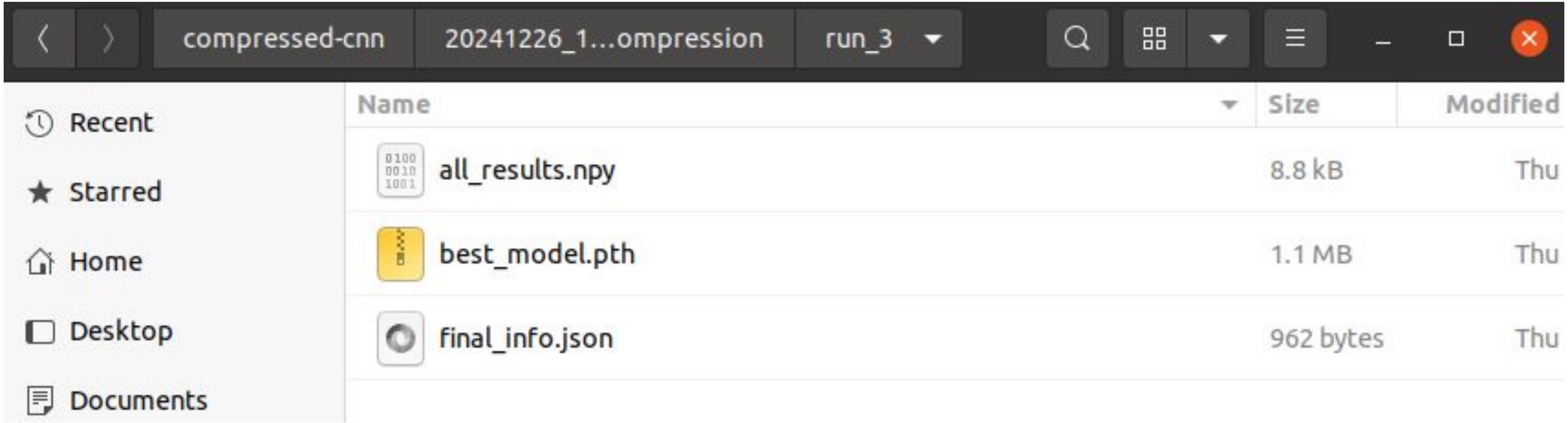
Experiment sandboxes I



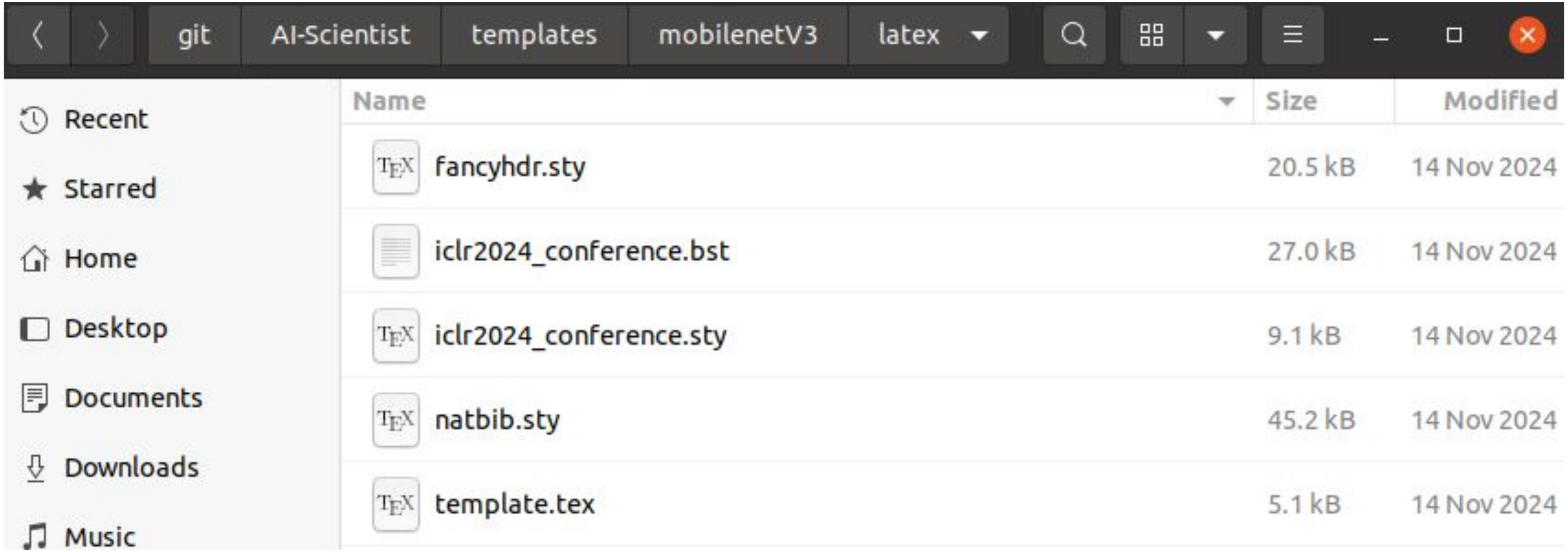
Experiment sandboxes II



Experiment sandboxes III



Latex template - ICLR 2024



Latex template - template.tex

Terminal

```
\begin{document}

\maketitle

\begin{abstract}
ABSTRACT HERE
\end{abstract}

\section{Introduction}
\label{sec:intro}
INTRO HERE

\section{Related Work}
\label{sec:related}
RELATED WORK HERE

\section{Background}
\label{sec:background}
BACKGROUND HERE

\section{Method}
\label{sec:method}
                                                        128,1         75%
```

Latex write-up

Open

launch_scientist.py
~/git/AI-Scientist

```python
import argparse
import json
import multiprocessing
import openai
import os
import os.path as osp
import shutil
import sys
import time
import torch
from aider.coders import Coder
from aider.io import InputOutput
from aider.models import Model
from datetime import datetime

from ai_scientist.generate_ideas import generate_ideas, check_idea_novelty
from ai_scientist.llm import create_client, AVAILABLE_LLMS
from ai_scientist.perform_experiments import perform_experiments
from ai_scientist.perform_review import perform_review, load_paper, perform_improvement
from ai_scientist.perform_writeup import perform_writeup, generate_latex
```

Latex write-up - prompts

Open | perform_writeup.py | Save

~/git/AI-Scientist/ai_scientist

```python

per_section_tips = {
    "Abstract": """
- TL;DR of the paper
- What are we trying to do and why is it relevant?
- Why is this hard?
- How do we solve it (i.e. our contribution!)
- How do we verify that we solved it (e.g. Experiments and results)

Please make sure the abstract reads smoothly and is well-motivated. This should be one continuous paragraph with no breaks between the lines.
""",
    "Introduction": """
- Longer version of the Abstract, i.e. of the entire paper
- What are we trying to do and why is it relevant?
- Why is this hard?
- How do we solve it (i.e. our contribution!)
- How do we verify that we solved it (e.g. Experiments and results)
- New trend: specifically list your contributions as bullet points
- Extra space? Future work!
""",
    "Related Work": """
- Academic siblings of our work, i.e. alternative attempts in literature at trying to
```

The AI-Scientist

Towards Fully Automated Open-Ended Scientific Discovery