Generative-AI Powered Inference with gpi-pack

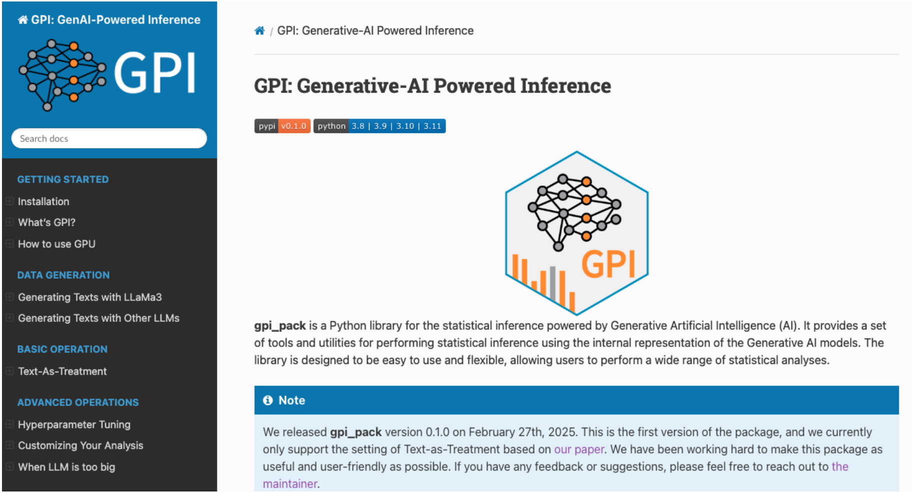
Kentaro Nakamura

Harvard University

Oxford Computational Political Science Group
May 20, 2025

Joint work with Kosuke Imai

gpi-pack: https://gpi-pack.github.io/



GPI: GenAI-Powered Inference

GPI

Search docs

GETTING STARTED

- Installation
- What's GPI?
- How to use GPU

DATA GENERATION

- Generating Texts with LLaMa3
- Generating Texts with Other LLMs

BASIC OPERATION

- Text-As-Treatment

ADVANCED OPERATIONS

- Hyperparameter Tuning
- Customizing Your Analysis
- When LLM is too big

/ GPI: Generative-AI Powered Inference

GPI: Generative-AI Powered Inference

pypi v0.1.0 python 3.8 | 3.9 | 3.10 | 3.11

GPI

gpi_pack is a Python library for the statistical inference powered by Generative Artificial Intelligence (AI). It provides a set of tools and utilities for performing statistical inference using the internal representation of the Generative AI models. The library is designed to be easy to use and flexible, allowing users to perform a wide range of statistical analyses.

Note

We released **gpi_pack** version 0.1.0 on February 27th, 2025. This is the first version of the package, and we currently only support the setting of Text-as-Treatment based on our paper. We have been working hard to make this package as useful and user-friendly as possible. If you have any feedback or suggestions, please feel free to reach out to the maintainer.

Computing Environment: How to use GPU

- `gpi-pack` is built upon PyTorch that supports GPU acceleration
- GPU is not essential, but it significantly speeds up computation

- Two ways to use GPU
  1. Local GPU Computing
  2. Cloud GPU computing platforms (e.g., AWS, Google Colab)
- Check if GPU is available:

```
import torch
print(torch.cuda.is_available())
```

Google Colab: https://colab.research.google.com/

- Free, cloud Jupyter notebook environment offering access to GPUs

- **How to use**:
    1. Create a notebook / Upload notebook to Google Drive
    2. Go to Runtime and Change Runtime Type
    3. Select GPU as the hardware accelerator

- Free tier has limits (e.g., 12 hours per session)
- Colab Pro/Pro+ offers more compute and access to premium GPUs

Three Steps of Implementation

1. Generating internal representation with Generative-AI
   - Text: Open-Source LLM (e.g., LLaMa, Gemma)
   - Image: Diffusion model (e.g., Stable Diffusion)
2. Hyperparameter tuning of nuisance functions for the outcome model
3. Estimate causal effects

STEP1: Generating Internal Representation with GenAI

- Two prompting strategies
  1. Create: create texts / images from scratch
  2. Reuse: repeating the same texts and images as the inputs

- Two options to use deep generative models
  1. Deploying models locally
     - Requires GPU, computationally heavy
     - Impossible to deploy the heavy model (e.g., LLaMa-495B)
     - Model quantization often helps
  2. Using API that allows users to access internal states
     - National Deep Inference Fabric (NDIF) offers the package `nnsight`
     - Website: https://nnsight.net/
     - Plan to release the functionality for `nnsight` soon

Example: Deploying models locally (1)

- Find your favorite model from `https://huggingface.co/`
  - Typically, the documentation shows how to deploy the model

```python
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

checkpoint = 'meta-llama/Meta-Llama-3.1-8B-Instruct'

tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForCausalLM.from_pretrained(
    checkpoint,
    device_map="auto",
    torch_dtype=torch.float16
)
```

Example: Deploying models locally (2)

- Use `extract_and_save_hidden_states` function in `gpi_pack`

```python
from gpi_pack.llm import extract_and_save_hidden_states

prompts = [
    'Create a biography of a politician named John Doe',
    'Create a biography of a politician named Jane Smith',
]

extract_and_save_hidden_states(
    prompts = prompts,
    output_hidden_dir = <YOUR HIDDEN DIR>,
    save_name = <YOUR SAVE NAME>,
    tokenizer = tokenizer,
    model = model,
    task_type = "create"
)
```

Example: Deploying models locally (3)

- To repeat the input texts, set `task_type == "repeat"`

```python
from gpi_pack.llm import extract_and_save_hidden_states

prompts = [
    'The Airports Commission, an independent body established...',
    'History show us that most large infrastructure projects...',
]

extract_and_save_hidden_states(
    prompts = prompts,
    output_hidden_dir = <YOUR HIDDEN DIR>,
    save_name = <YOUR SAVE NAME>,
    tokenizer = tokenizer,
    model = model,
    task_type = "repeat"
)
```

STEP2: Hyperparameter Tuning

- The performance of the estimator is sensitive to hyperparameters
  - Fitting of deconfounder is especially important

- Hyperparameters
  - learning rate
  - batch size
  - dropout rate
  - size and width

- `Optuna` offers efficient hyperparameter optimization
  - `gpi_pack` offers the class `TarNetHyperparameterTuner`
  - Only need to specify data and the choice of hyperparameters

Example: Hyperparameter Tuning with Optuna

```python
from gpi_pack.TarNet import TarNetHyperparameterTuner
import optuna

obj = TarNetHyperparameterTuner(
    # Data
    T = df['TreatmentVar'].values,
    Y = df['OutcomeVar'].values,
    R = hidden_states,
    # Hyperparameters
    learning_rate = [1e-4, 1e-5],
    dropout = [0.1, 0.2],
    architecture_y = ["[200, 1]", "[100,1]"],
    architecture_z = ["[1024]", "[2048]"]
)

# Hyperparameter tuning with Optuna
study = optuna.create_study(direction='minimize')
study.optimize(obj.objective, n_trials=100)
```

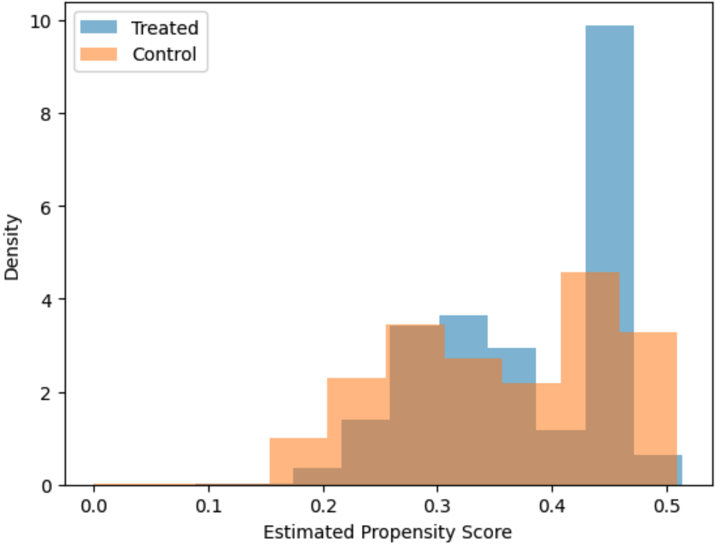
STEP3: Estimate Treatment Effect

- K-fold cross-fitting to estimate the treatment effect
- `estimate_ate_k` function deals with all the procedures
  - Propensity score function needs to be Lipschitz continuous
  - Default choice is neural network with spectral normalization
- You should always plot the estimated propensity score distributions
  - For text/image-as-treatment, it works as a diagnosis of separability assumption

Example: Estimate Treatment Effect

```python
# estimate treatment effects
ate, se = estimate_k_ate(
     R= hidden_states,
     Y= df['OutcomeVar'].values,
     T= df['TreatmentVar'].values,
     K=2, #K-fold cross-fitting
     lr = 2e-5, #learning rate
     architecture_y = [200, 1], #outcome model architecture
     architecture_z = [2048], #deconfounder architecture
     plot_propensity = True, #visualize propensity scores
)
```

Example: Propensity Score Distributions



Conclusion

- gpi-pack is an open-source software to implement GPI
- Website: https://gpi-pack.github.io/
- If you have any question, find bugs, or have any suggestions,
  1. Please open an issue on GitHub (https://github.com/gpi-pack/gpi_pack)
  2. Email me at knakamura [at] g.harvard.edu