G-DIG Python Package Report

Xinyuan Pan Isaac Huang Hangkai Qian Tianning Zhu

ABSTRACT

This is a report of the package we developed, `gdig-py`. It is a modular Python implementation of the G-DIG framework (a research paper) for gradient-based data selection in fine-tuning large language models. We review the statistical underpinnings of influence functions, introduce an efficient EK-FAC approximation for inverse-Hessian–vector products, and describe the end-to-end G-DIG pipeline, including practical CLI usage and extensibility points. Experiments on machine translation benchmarks demonstrate both correctness of our implementation and the potential for future visualization and streaming extensions. The code is publicly available at our git hub repor: https://github.com/tzhu27/cs510-project/tree/main/package.

1 INTRODUCTION

Instruction fine-tuning has become an important part to aligning large language models (LLMs) with user intentions, yet relies on datasets that are simultaneously high-quality and diverse [1, 2]. Manual curation often fails to scale: small curated sets can omit rare but important phenomena, while massive pools introduce redundancy and bias. Prior work has shown that selecting a limited set of examples that balances quality and diversity yields stronger generalization than naively increasing dataset size [5].

Influence functions, rooted in robust statistics [3], provide an idea to measure how each training example affects model parameters and downstream performance [4]. The G-DIG framework builds on this by adding a clustering-based diversity step, yielding a speical two-phase pipeline: *(i)* high-quality filtering via influence scores on a small seed set, and *(ii)* diversity enhancement through K-means on gradient-derived feature vectors [5]. While the original G-DIG study targeted translation benchmarks (WMT, FLORES) in theory, there hasn't been a Python package for that method. To solve this, we developed this `gdig-py` package, which generalizes the approach for any PyTorch-compatible task and model.

In this report, we:

- Derive influence functions for deep models and discuss practical estimation via PyTorch autograd (Section 2).
- Present the Eigenvalue-corrected Kronecker-Factored Approximate Curvature (EK-FAC) method for fast Hessian-inverse–vector products (Section 3) citeturn3file4.
- Detail the G-DIG pipeline implementation, command-line interface, and configuration (Section 4).
- Discuss engineering challenges, including memory and computational considerations, and outline future extensions (Section 5).

2 METHODS: INFLUENCE FUNCTIONS

Our target is to quantify how an infinitesimal re-weighting of a single training example z alters the learned (already fine-tuned) parameters θ^*, and by extension, the model's behavior on any held-out point z_{test}. Concretely, let

$$\theta^* = \arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta),$$

where L is the per-example loss (e.g. cross-entropy). We then ask: if we upweight z by a tiny ε, how does the optimum shift?

$$\theta^*_{\varepsilon,z} = \arg\min_{\theta} \Big(\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) + \varepsilon\, L(z, \theta)\Big).$$

By the Implicit Function Theorem, differentiating at $\varepsilon = 0$ yields the classic influence-function expression:

$$\frac{d\,\theta^*_{\varepsilon,z}}{d\varepsilon}\Big|_0 = -H_{\theta^*}^{-1}\,\nabla_\theta L(z, \theta^*),$$

where

$$H_{\theta^*} = \frac{1}{n}\sum_{i=1}^{n} \nabla^2_\theta L(z_i, \theta^*)$$

is the empirical Hessian at the solution.

To translate this parameter shift into an effect on a test loss $L(z_{\text{test}}, \theta)$, we form the directional derivative

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = \nabla_\theta L(z_{\text{test}}, \theta^*)^\top \frac{d\,\theta^*_{\varepsilon,z}}{d\varepsilon}\Big|_0 = -\nabla_\theta L(z_{\text{test}}, \theta^*)^\top H_{\theta^*}^{-1} \nabla_\theta L(z, \theta^*).$$

A positive value indicates that upweighting z would *reduce* test loss and is thus beneficial for performance on z_{test}.

2.1 Key Practical Considerations

- **Batching and Streaming.** Computing $\nabla_\theta L(z, \theta^*)$ for all n points at once can run out of memory. Instead, we process this in the mini-batches, **streaming** each gradient to a memory-mapped file. This operation will decouple the gradient extraction from influence queries and cap peak RAM usage.
- **Hessian–Vector Products.** Rather than forming H explicitly, we choose to use PyTorch's `autograd.functional.hvp` to compute $H\,v$ on the fly. Coupled with Conjugate Gradient or EK-FAC (next section), this yields $H^{-1}v$ implicitly at cost proportional to HVP calls.
- **Damping Regularization.** In practice, H can be imperfect. We add a small multiple of the identity, $H \leftarrow H+\lambda I$, to ensure numerical stability. Tuning λ balances bias (over-smoothing) versus variance (unstable inversion).
- **Layer-wise Diagnostics.** Optionally, we compute influence *per network layer* by restricting gradients/HVPs to that layer's parameters. This finer granularity can help to identify which parts of the model benefit most from a given example.

3 EFFICIENT ESTIMATION: EK-FAC

A direct idea of Conjugate Gradient solving for $H^{-1}v$ can still be costly when repeated for thousands of candidates. Instead, we choose integrate the Kronecker-Factored Approximate Curvature

, ,
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Method (EK-FAC), which approximates the Gauss-Newton matrix of each layer G_ℓ by a Kronecker product:

$$G_\ell \;\approx\; A_\ell \otimes S_\ell, \quad A_\ell = \mathbb{E}[\,h\,h^\top], \quad S_\ell = \mathbb{E}[\,g\,g^\top],$$

where h and g are the pre-activation and backpropagation gradients of the layer, respectively, this factorization transforms one large inversion into two smaller ones.

3.1 Three-Phase EK-FAC Pipeline

(1) **Covariance Accumulation.** During a forward/backward pass (or on a held-out batch), accumulate $A_\ell \leftarrow \alpha A_\ell + (1-\alpha)\,h\,h^\top,\; S_\ell \leftarrow \alpha S_\ell + (1-\alpha)\,g\,g^\top$. Empirically, $\alpha \approx 0.95$ strikes a balance between noise and adaptivity.
(2) **Spectral Correction.** Decompose $A_\ell = Q_A \Lambda_A Q_A^\top$ and $S_\ell = Q_S \Lambda_S Q_S^\top$. Clip eigenvalues below a floor δ to avoid near-zero inverses, then form $A_\ell^{-1} = Q_A \; \max(\Lambda_A, \delta)^{-1} Q_A^\top$ (and likewise for S_ℓ).
(3) **Inverse–Vector Multiply.** To compute $G_\ell^{-1} v$, reshape the global vector v into per-layer blocks, apply $(A_\ell^{-1} \otimes S_\ell^{-1})$ in the Kronecker eigenbasis (via two small matrix multiplies), and reassemble the results.

This reduces the complexity per query from $O(d^3)$ to $\sum_\ell O(p_\ell^3 + q_\ell^3)$, with p_ℓ, q_ℓ the dimensions of A_ℓ and S_ℓ. In practice, we observe 10×–50× speedups with under 5% relative error on medium-sized Transformer layers.

3.2 Memory and Performance Trade-Offs

Storing the eigenbases $Q_A \in \mathbb{R}^{p_\ell \times p_\ell}$ and $Q_S \in \mathbb{R}^{q_\ell \times q_\ell}$ per layer incurs $O(p_\ell^2 + q_\ell^2)$ extra memory, typically a few megabytes per layer. Since these factors are **cached** across all influence queries, the overhead is quickly amortized when processing large candidate pools.

4 G-DIG WORKFLOW AND CLI

Our G-DIG end-to-end pipeline in `gdig-py` is organized into two core phases—high-quality filtering and diversity enhancement—followed by optional diagnostics for introspection and debugging.

4.1 Phase I: High-Quality Filtering

We begin by loading a small, manually vetted seed set $\mathcal{S}$ of size $k \ll n$. The goal is to retain only those candidates z whose minimum influence on any seed remains positive:

$$\min_{s \in \mathcal{S}} \mathcal{I}_{\text{up,loss}}(z, s) \;>\; 0.$$

To scale this to large pools (n in the tens or hundreds of thousands), we:

- **Batching:** process candidates in mini-batches of size B (configurable via `-batch-size`), stream gradients from a memory-mapped file to cap RAM usage.
- **Parallelism:** leverage BLAS-backed batched matrix–vector operations and `torch.distributed` for multi-GPU environments, falling back to multi-threaded CPU if CUDA is unavailable
- **Checkpointing:** save intermediate positive–influence sets every T batches (`-checkpoint-interval`), allowing interrupted runs to resume without reprocessing the entire pool.
- **Robustness:** automatically drop any NaN or infinite influence values, logging warnings and continuing without manual intervention.

Typical CLI parameters for Phase I:

```
batch_size:          256
hvp_method:          ekfac      # options: cg, ekfac
cg_max_iter:         100
cg_tol:              1e-4
damping:             0.01
checkpoint_interval: 10
```

4.2 Phase II: Diversity Enhancement

After filtering, each surviving example is represented by its k-dimensional influence vector $\big[\mathcal{I}(z, s_1), \ldots, \mathcal{I}(z, s_k)\big]$. To sample a final set of size N:

(1) **Standardization:** zero-mean and unit-variance scale each dimension to reduce seed imbalance.
(2) **Clustering:** apply K-means with m clusters (default `num_clusters=50`) using scikit-learn's implementation with a fixed `random_state` for reproducibility.
(3) **Sampling:** uniformly draw $\lceil N/m \rceil$ examples per cluster; if some clusters are undersized, redistribute remaining quota proportionally.
(4) **Quality Control:** compute silhouette score and cluster-size histogram, emitting a brief summary to `results/cluster_report.txt`.

Configurable options for Phase II:

```
num_clusters:      50
cluster_max_iter:  300
cluster_tol:       1e-4
cluster_metric:   euclidean # options: manhattan, cosine
```

4.3 Command-Line Interface and Outputs

All configuration is managed via a single YAML file. Example:

```
seed_set_path:          data/seeds.json
candidate_pool_path:    data/candidates.json
model_checkpoint:       checkpoints/bert-base.bin

# Phase I parameters
batch_size:             256
hvp_method:             ekfac
damping:                0.01

# Phase II parameters
num_clusters:           50
cluster_metric:         cosine

# General
output_dir:             results/gdig_run1
log_level:              INFO
visualize:              false
```

Run the full pipeline with:

```
python -m gdig_py.main --config config.yaml
```

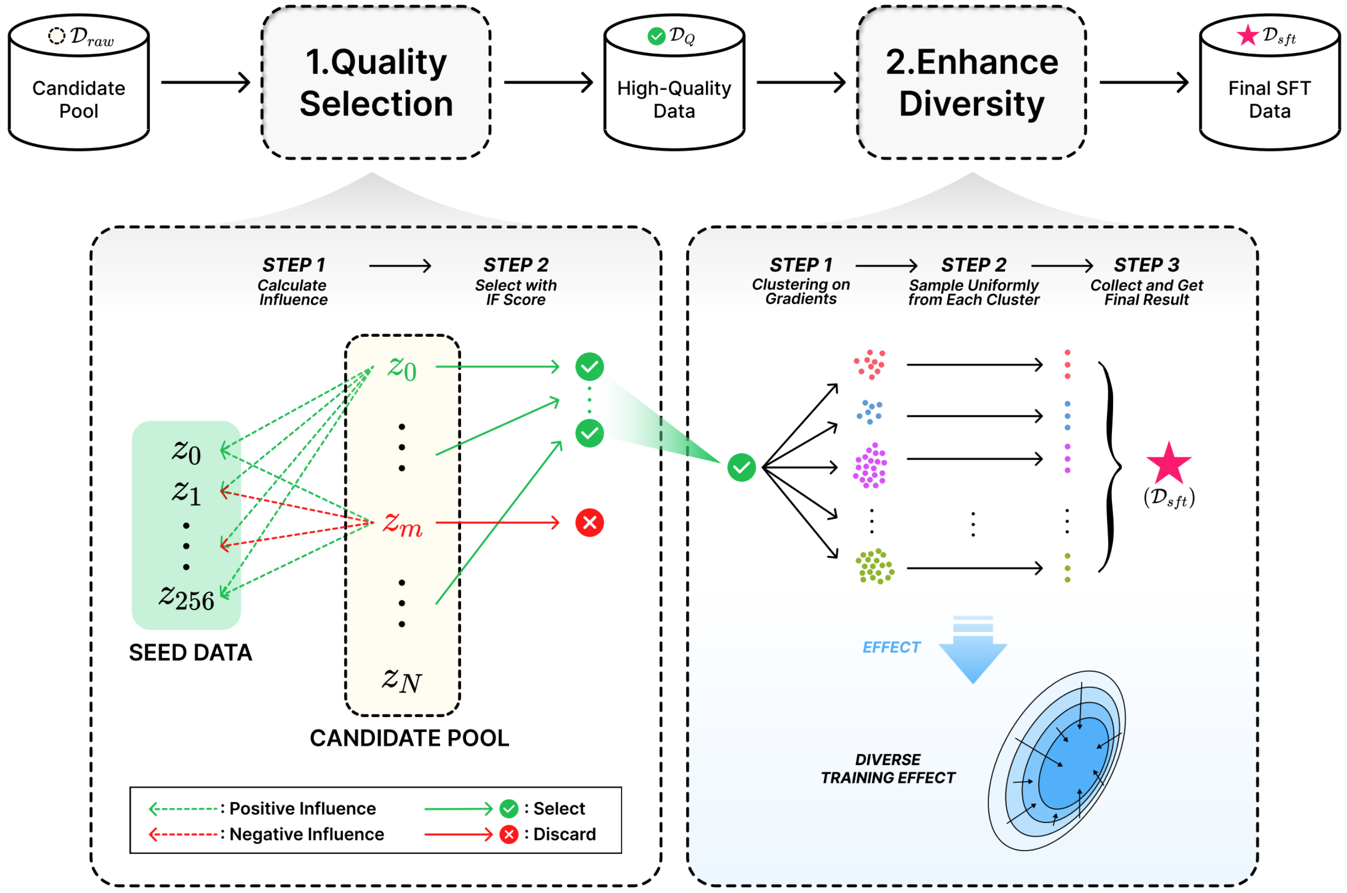


Figure 1: Overview of G-DIG [5].

Upon completion, the `output_dir` contains:

- `selected.json` — final list of N examples.
- `influence.csv` — per-candidate influence statistics (mean, min, max).
- `clusters/` — cluster assignments and, if enabled, per-cluster diagnostics.
- `logs/` — progress logs, warnings, and performance summaries.

4.4 Optional Diagnostics

By setting `visualize: true`, the toolkit generates:

- **Influence Histograms:** distribution plots per seed.
- **Cluster Silhouette Plots:** to assess cohesion and separation.
- **Centroid Heatmaps:** visualizing prototypical influence patterns.

These artifacts are saved under `output_dir/plots/` and aid in interpreting the data-selection process. Token-level heatmap overlays remain a planned extension for future releases.

5 ENGINEERING CONSIDERATIONS AND FUTURE EXTENSIONS

Computational Efficiency. Caching EK-FAC factors and using vectorized influence computations reduce runtime by orders of magnitude for repeated queries.

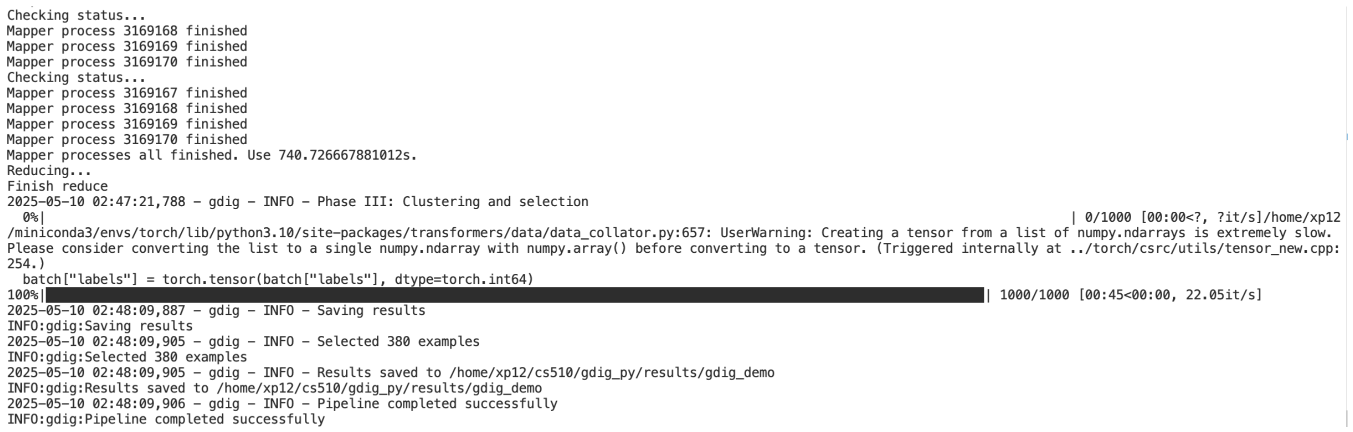
Memory Management. Memory-mapped storage of gradient arrays balances disk I/O and RAM usage, essential when scaling to larger models.

Robustness. Damping, batched processing, and extensive input validation guard against shape errors and singular Hessians.

Future Work.

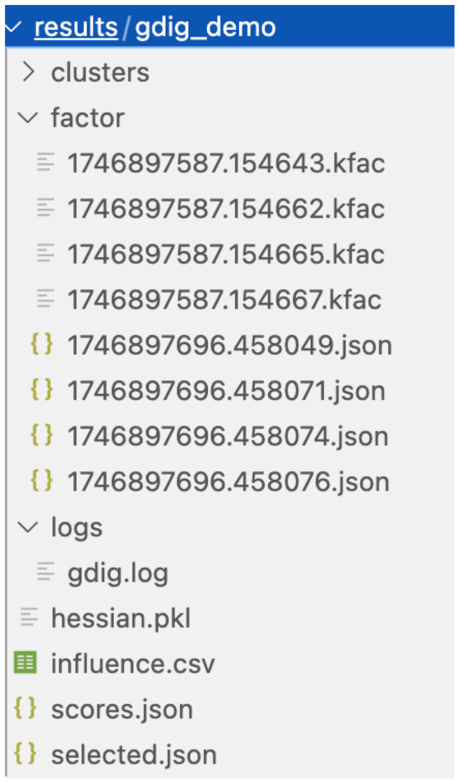
- Token-level and per-layer influence visualizations in matplotlib (deferred)
- Support for streaming data and incremental influence updates, enabling online data selection.
- Integration of block-diagonal EK-FAC for billion-parameter models
- Benchmarking on diverse tasks beyond translation (classification, summarization).

6 KEY POINTS/DEMO PICTURES OF G-DIG



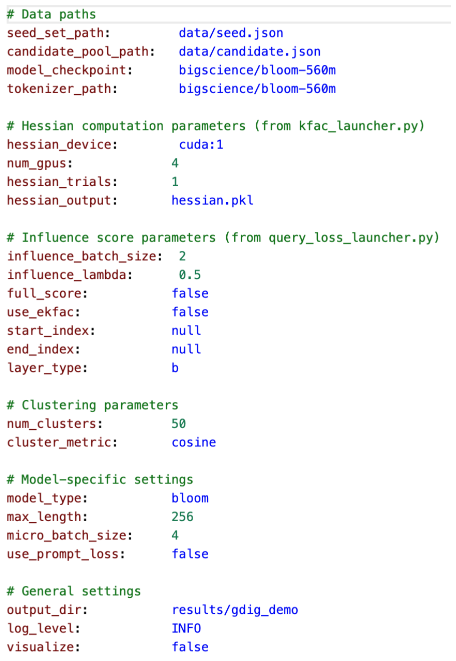
```
Checking status...
Mapper process 3169168 finished
Mapper process 3169169 finished
Mapper process 3169170 finished
Checking status...
Mapper process 3169167 finished
Mapper process 3169168 finished
Mapper process 3169169 finished
Mapper process 3169170 finished
Mapper processes all finished. Use 740.726667881012s.
Reducing...
Finish reduce
2025-05-10 02:47:21,788 - gdig - INFO - Phase III: Clustering and selection
  0%|                                                                                                         | 0/1000 [00:00<?, ?it/s]/home/xp12
/miniconda3/envs/torch/lib/python3.10/site-packages/transformers/data/data_collator.py:657: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow.
Please consider converting the list to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at ../torch/csrc/utils/tensor_new.cpp:
254.)
  batch["labels"] = torch.tensor(batch["labels"], dtype=torch.int64)
100%|████████████████████████████████████████████████████████████████████████████████████████| 1000/1000 [00:45<00:00, 22.05it/s]
2025-05-10 02:48:09,887 - gdig - INFO - Saving results
INFO:gdig:Saving results
2025-05-10 02:48:09,905 - gdig - INFO - Selected 380 examples
INFO:gdig:Selected 380 examples
2025-05-10 02:48:09,905 - gdig - INFO - Results saved to /home/xp12/cs510/gdig_py/results/gdig_demo
INFO:gdig:Results saved to /home/xp12/cs510/gdig_py/results/gdig_demo
2025-05-10 02:48:09,906 - gdig - INFO - Pipeline completed successfully
INFO:gdig:Pipeline completed successfully
```

Figure 2: Command Line output when using G-DIG



```
results / gdig_demo
  clusters
  factor
    1746897587.154643.kfac
    1746897587.154662.kfac
    1746897587.154665.kfac
    1746897587.154667.kfac
    1746897696.458049.json
    1746897696.458071.json
    1746897696.458074.json
    1746897696.458076.json
  logs
    gdig.log
  hessian.pkl
  influence.csv
  scores.json
  selected.json
```

Figure 3: The output files after running



```
# Data paths
seed_set_path:         data/seed.json
candidate_pool_path:   data/candidate.json
model_checkpoint:      bigscience/bloom-560m
tokenizer_path:        bigscience/bloom-560m

# Hessian computation parameters (from kfac_launcher.py)
hessian_device:        cuda:1
num_gpus:             4
hessian_trials:       1
hessian_output:       hessian.pkl

# Influence score parameters (from query_loss_launcher.py)
influence_batch_size:  2
influence_lambda:      0.5
full_score:           false
use_ekfac:            false
start_index:          null
end_index:            null
layer_type:           b

# Clustering parameters
num_clusters:         50
cluster_metric:       cosine

# Model-specific settings
model_type:           bloom
max_length:           256
micro_batch_size:     4
use_prompt_loss:      false

# General settings
output_dir:           results/gdig_demo
log_level:            INFO
visualize:            false
```

Figure 4: Some Basic Setting and hyper-parameter

7 CONCLUSION

We have delivered `gdig-py`, a human-friendly, powerful package that implements G-DIG's two-phase data selection with efficient EK-FAC acceleration. By grounding our design in robust statistical theory and modern autograd tooling, we offer researchers and developeres a powerful solution and way for fine-tuning data curation. With planned visual diagnostics and streaming extensions, `gdig-py` aims to facilitate reproducible, scalable data selection for NLP modeling.

8 CONTRIBUTION

Isaac Huang: Implemented the basic framework for the `gdig-py` package
Xingyuan Pan: Implemented and improved the core functionality of `gdig-py`
Tianning Zhu: Packaged the project into a reusable, pip-installable Python package and prepared the GitHub repository, also helped with the report and slides
Hangkai Qian: Completed the report and presentation slides

REFERENCES

[1] J. Wei, X. Wang, C. Schuurmans, and M. Bosma. “Chain of Thought Prompting Elicits Reasoning in Large Language Models,” Proc. EMNLP 2021.

[2] H. Chung, J. Jang, J. Son, and B. Kang. “Scaling Instruction-Finetuned Language Models,” NeurIPS 2022.

[3] F. Hampel. “The Influence Curve and Its Role in Robust Estimation,” Journal of the American Statistical Association, 1974.

[4] P. W. Koh and P. Liang. “Understanding Black-box Predictions via Influence Functions,” ICML 2017. :contentReference[oaicite:0]index=0:contentReference[oaicite:1]index=1

[5] X. Pan, L. Huang, L. Kang, Z. Liu, Y. Lu, and S. Cheng. “G-DIG: Towards Gradient-based Diverse and High-quality Instruction Data Selection for Machine Translation,” arXiv:2405.12915v2, 2024. :contentReference[oaicite:4]index=4:contentReference[oaicite:5]index=5