

HOMework 7^{*}

DATABASES AND CODING ASSISTANTS[†]

24-880 AI AGENTS FOR ENGINEERS
<https://cmuagents.github.io>

OUT: Sunday, March 15th 2026

DUE: Sunday, March 22nd 2026

TAs: Peter Pak

CAs: Hardik Choudhary

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template for the submissions which can be written in L^AT_EX (via [Overleaf](#)) or directly on the PDF document. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Code:** Gradescope's Autograder will be used to evaluate the coding portions of this assignment. In order to best obtain the points for this assignment, please follow the code upload instructions. If you do not follow the instructions correctly the autograder may not work properly.

Question	Points
FAISS (Programming)	45
TDFlow (Written)	20
TDFlow (Programming)	30
Programming Scripts Upload	0
Collaboration and Generative AI Questions	5
Total:	100

^{*}Adapted from CMU 10-623 Generative AI Homework Template

[†]Compiled on Saturday 14th March, 2026 at 20:39

Introduction

In this homework assignment we'll look into how similarity search works at scale for databases and how coding assistants such as TDFlow work.

At a high-level, this homework will cover the following aspects.

1. Explore some of the concepts and functions within the FAISS library.
2. Understand how the agentic system for TDFlow works.
3. Implement aspects of TDFlow.

Local Environment

For this homework we'll be using `uv` as the project and package manager. If you haven't install `uv`, you can follow the installation instructions here <https://docs.astral.sh/uv/>. You can explore all of the available commands using `uv --help` but in short, here are a couple of useful commands that you'll often use:

<code>source .venv/bin/activate</code>	(Linux and Mac) Activates python environment.
<code>source .venv/Scripts/Activate</code>	(Windows) Activates python environment.
<code>uv sync</code>	Installs / synchronizes projects dependencies.
<code>uv add <package></code>	Installs <code>pip</code> package and adds it to project dependencies.
<code>uv remove <package></code>	Removes <code>pip</code> package dependency from project.
<code>uv run <script.py></code>	Runs python script inside of <code>uv</code> environment.

1. Navigate to folder project directory and synchronize dependencies with:

```
uv sync
```

2. Activate the environment

Mac and Linux:

```
source .venv/bin/activate
```

Windows:

```
source .venv/Scripts/activate
```

3. Test `uv` works and run `main.py` file.

```
uv run main.py
```

1 FAISS (Programming) (45 points)

Similarity search relies on the computation of a k -NN graph where it provides a directed graph that utilizes each vector in the database as a node and each edge is connected to its k nearest neighbors (Johnson et al. [2017]). However, the computation of the k -NN graph is a computationally expensive process which becomes exponentially worse with data represented using increasing dimensionality (Weber et al. [1998]). Executing an exhaustive search for large scale data becomes an impractical task using this inefficient approach and instead leveraging the parallel compute efficiency of GPU's offers an approach that can more efficiently perform similarity search albeit with a small tradeoff in accuracy.

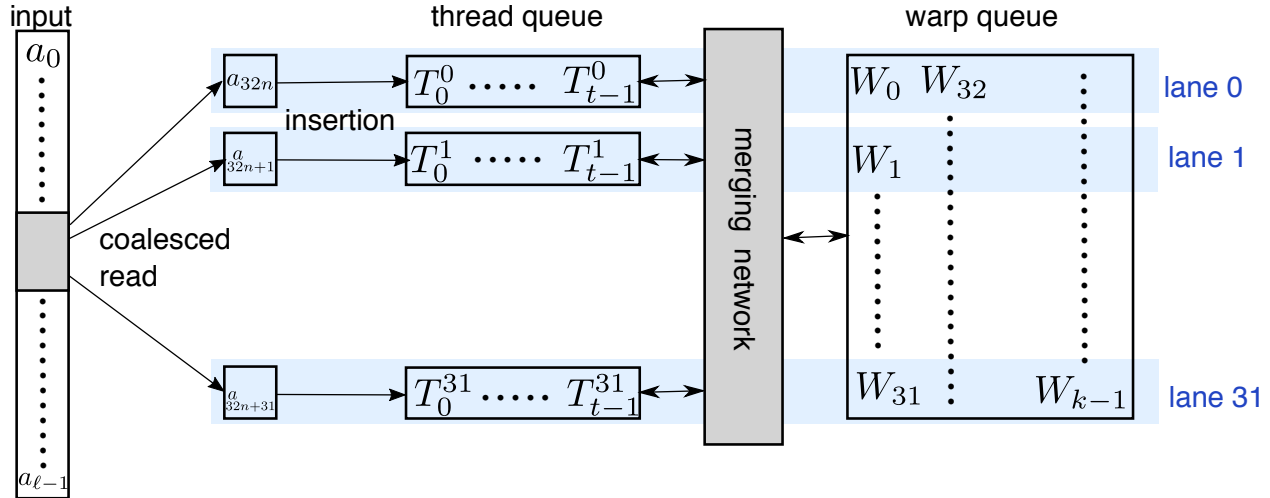


Figure 1: From Johnson et al. [2017], “Overview of WarpSelect. The input values stream in on the left, and the warp queue on the right holds the output result.”

With the usage of GPUs, search can be performed in batches allowing for greater efficiency when compared to CPU based methods (Johnson et al. [2017]). In addition a compressed approach is taken for the domain search utilizing the Inverted File with Asymmetric Distance Computation (IVFADC) indexing system outlined by Jégou et al. [2011] for a non-exhaustive search scheme. By pruning the search space to only the k nearest neighbor and performing quantization based compression, this highly parallelizable approach drastically reduces the number of scans and expedites the similarity search task (Johnson et al. [2017]). This methodology, Facebook AI Similarity Search (FAISS) by Douze et al. [2024], has been implemented in commercial vector database products such as Pinecone, ChromaDB, and etc.

- 1.1. In `programming/similarity_search.py`, implement the following functions using the `faiss` library.
- 1.1.a. (3 points) `create_flat_l2_index(d)` — Create a flat L2 (Euclidean distance) index with dimension `d`.
 - 1.1.b. (3 points) `create_flat_ip_index(d)` — Create a flat inner product index with dimension `d`.
 - 1.1.c. (3 points) `get_index_dimension(index)` — Return the dimensionality of the index.
 - 1.1.d. (3 points) `is_index_trained(index)` — Return `True` if the index is trained and ready for use.
 - 1.1.e. (3 points) `add_vectors(index, vectors)` — Add a NumPy array of vectors (shape: $n \times d$, dtype: `float32`) to the index.
 - 1.1.f. (3 points) `get_index_size(index)` — Return the number of vectors currently stored in the index.
 - 1.1.g. (3 points) `search_index(index, query_vectors, k)` — Search for `k` nearest neighbors for each query vector. Return `(distances, indices)`, each of shape $(n_queries, k)$.
 - 1.1.h. (3 points) `find_nearest_neighbor(index, query_vector)` — Return the integer index of the single nearest neighbor for a query vector of shape $(1, d)$.
 - 1.1.i. (3 points) `normalize_vectors(vectors)` — Normalize vectors in-place using L2 normalization. Return the normalized vectors.
 - 1.1.j. (3 points) `build_l2_index(d, vectors)` — Create a flat L2 index with dimension `d`, add vectors to it, and return the populated index.
 - 1.1.k. (3 points) `get_nearest_distances(index, query_vector, k)` — Return a 1D array of the distances to the `k` nearest neighbors for a single query vector.
 - 1.1.l. (3 points) `exact_match_distance(index, vector)` — Search the index for the given vector and return the distance to its nearest neighbor as a `float`. If the vector exists in the index, this should be approximately `0.0`.
 - 1.1.m. (3 points) `create_ivf_index(d, nlist)` — Create an untrained IVF flat index with dimension `d` and `nlist` clusters using L2 metric. *Hint: you will need a quantizer (`IndexFlatL2`) and `faiss.METRIC_L2`.*
 - 1.1.n. (3 points) `save_index(index, filepath)` — Save a FAISS index to a file at the given path.
 - 1.1.o. (3 points) `load_index(filepath)` — Load and return a FAISS index from a file at the given path.

2 TDFlow (Written) (20 points)

TDFlow by Han et al. [2026] is an agentic workflow that enables a Test Driven Development (TDD) by George and Williams [2004] approach to software engineering. In this work, the authors surmise this workflow can help streamline the TDD workflow by allowing the developers to design and construct the necessary tests and relying on the large language model to generate the necessary code. This system utilizes subagents to generate tests, explore files, revise / patch software, and debug executed tests. A notable distinction in this approach is that the next course of action follows a structured path rather than allowing the large language model to dictate the workflow. The subagents then execute the necessary tool calls in order to successfully complete each task. TDFlow achieves a score of 88.8% on SWE-Bench Lite in contrast to alternative methods such as OpenHands (47.8%) Wang et al. [2025], ExpeRepair (48.6%) Mu et al. [2025], SWE-Agent (49.0%) Yang et al. [2024], and Agentless (61.0%) Xia et al. [2024].

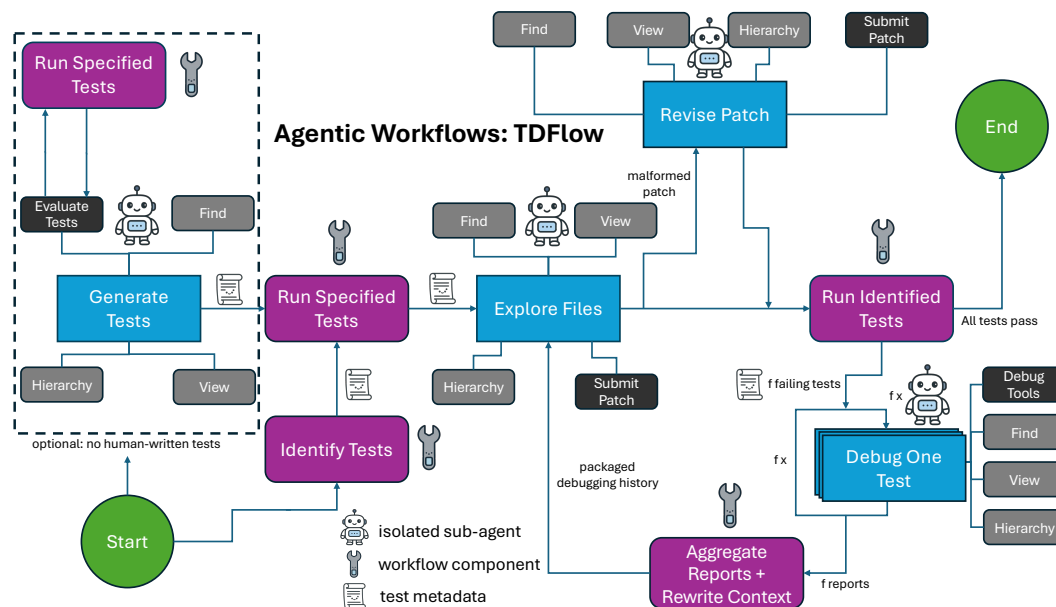


Figure 2: From Han et al. [2026], “The workflow behind Test-Driven Flow (TDFlow). TDFlow is a purely test driven, agentic **workflow** for resolving repository scale issues. The entrypoint to TDFlow begins with either human-written reproduction tests or, optionally, to have TDFlow generate reproduction tests. Afterwards, the tests are run and provided to the *Explore Files* LLM sub-agent with the sole task of exploring the repository in order to propose a patch. The tests are run on the proposed patch before the *Debug One* sub-agent debugs each failing test individually with a dedicated debugger tool and generates reports. Those reports are used by the *Explore Files* sub-agent to propose another patch.”

2.1. (5 points) Explain the role of **Generate Tests** within TDFlow

2.2. (5 points) What does **Explore Files** in TDFlow do?

2.3. (5 points) What is **Debug One**?

2.4. (5 points) How does **Revise Patch** work?

3 TDFlow (Programming) (30 points)

In `programming/tdflow.py`, implement a simplified version of the TDFlow agentic workflow. A `generate(prompt)` helper that calls a local language model is already provided. Use `generate()` inside each function below to produce the required output.

- 3.1. Implement the four sub-agents and the orchestration loop described below. Each sub-agent should construct an appropriate prompt, call `generate()`, and return the result in the specified type.

3.1.a. (6 points) **`generate_tests(issue)`**

Given a GitHub issue description, generate reproduction tests. Return a `list[str]` where each element is a test function string.

3.1.b. (6 points) **`explore_files(issue, failing_tests, debug_reports)`**

Given the issue, the failing test source strings, and any debug reports from a previous iteration (empty on the first call), propose a patch. Return the patch as a non-empty `str`.

3.1.c. (6 points) **`debug_one(test_source, test_message, issue, patch)`**

Given a single failing test's source code, its error message, the issue description, and the patch that was applied, produce a debug report. Return the report as a non-empty `str`.

3.1.d. (6 points) **`revise_patch(patch, error_message)`**

Given a malformed patch and the error produced when trying to apply it, return a revised patch as a non-empty `str` with corrected context lines. Do not alter the inserted code.

3.1.e. (6 points) **`tdflow(issue, tests, max_iterations)`**

Orchestrate the full simplified TDFlow loop:

1. If `tests` is `None` or empty, call `generate_tests` to produce them.
2. For up to `max_iterations`:
 - (a) Call `explore_files` to propose a patch.
 - (b) Call `revise_patch` to fix the patch if needed.
 - (c) Call `debug_one` for each test and collect the reports.
3. Return the final patch as a `str`.

4 Programming Scripts Upload (0 points)

To upload your code to Gradescope, run the following command to generate a zip file:

```
uv run code/zip_programming.py
```

Upload the generated zip file to the appropriate programming slot on Gradescope.

4.1. Make sure the following files are included when uploading to Gradescope

☐ `similarity_search.py`

☐ `tdflow.py`

5 Collaboration and Generative AI Questions (5 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 5.1. (2 points) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 5.2. (3 points) Did you use generative AI for parts of this assignment? If so, include the input prompts along with links to the conversation.

References

- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss library, 2024. URL <https://arxiv.org/abs/2401.08281>. Version Number: 4.
- Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337–342, April 2004. ISSN 0950-5849. doi: 10.1016/j.infsof.2003.09.011. URL <https://www.sciencedirect.com/science/article/pii/S0950584903002040>.
- Kevin Han, Siddharth Maddikayala, Tim Knappe, Om Patel, Austen Liao, and Amir Barati Farimani. TD-Flow: Agentic Workflows for Test Driven Development, January 2026. URL <http://arxiv.org/abs/2510.23761>. arXiv:2510.23761 [cs].
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs, 2017. URL <https://arxiv.org/abs/1702.08734>. Version Number: 1.
- Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011. ISSN 1939-3539. doi: 10.1109/TPAMI.2010.57. URL <https://ieeexplore.ieee.org/document/5432202>.
- Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. EXPEREPAIR: Dual-Memory Enhanced LLM-based Repository-Level Program Repair, June 2025. URL <http://arxiv.org/abs/2506.10484>. arXiv:2506.10484 [cs].
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents, April 2025. URL <http://arxiv.org/abs/2407.16741>. arXiv:2407.16741 [cs].
- R. Weber, H. Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. August 1998. URL <https://www.semanticscholar.org/paper/A-Quantitative-Analysis-and-Performance-Study-for-Weber-Schek/63eaeb0c48175065fffd096aad10aed712c6d7bbb>.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based Software Engineering Agents, October 2024. URL <http://arxiv.org/abs/2407.01489>. arXiv:2407.01489 [cs].
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, December 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.