

TEACHER RECOMMENDATION SYSTEM USING KNN ALGORITHM

ABSTRACT

The effectiveness of education heavily depends on the alignment between student needs and the teaching style of educators. In this paper, we present a teacher recommendation system utilizing the k-Nearest Neighbors (KNN) algorithm, implemented with CUDA (Compute Unified Device Architecture) for accelerated performance on GPU hardware. The recommendation system aims to assist educational institutions in matching teachers to students or courses based on various attributes such as expertise, student feedback, and course requirements. Traditional KNN algorithms, while effective, are computationally expensive when dealing with large datasets, which is a common challenge in modern educational systems. By leveraging the parallel processing capabilities of CUDA on GPUs, our implementation significantly reduces the time complexity of the KNN algorithm, enabling real-time recommendations even with extensive datasets. We evaluate the system on multiple datasets, demonstrating improved accuracy and efficiency. The experimental results reveal that GPU-accelerated KNN provides a scalable solution that outperforms conventional CPU-based implementations, offering a practical approach for large-scale teacher recommendation systems.

KEYWORDS: KNN(K-Nearest Neighbors), GPU(Graphics Processing Unit), CUDA (Compute Unified Device Architecture), Teacher Recommendation System.

INTRODUCTION

A Teacher Recommendation System aims to create the process of matching students with appropriate teachers based on student profiles and teacher performance data. Recommendation Systems were developed in order to address the accumulation of large volumes of diverse data in digital form, as a result of advances in Internet, web and cloud technologies. The need to store, process and analyze such data has arisen in various electronic fields. Here the goal of the project is to recommend the most suitable teachers for students based on various attributes such as student preferences, performance, teacher subject expertise, teaching methods, and ratings from

previous students. This system utilizes machine learning algorithms to enhance the learning experience by matching students with teachers who can address their learning needs effectively. The K-nearest-neighbors (KNN) algorithm is particularly useful in such a recommendation system because it classifies data points based on similarity. By analyzing student preferences, KNN can recommend the top k teachers who are closest to the student's profile. Here CUDA is used for parallelizing the code. The KNN algorithm involves calculating distances between the new student's data point and all other student data points in the dataset. This is computationally expensive for large datasets. CUDA allows this distance calculation to be performed in parallel on the GPU, reducing the computational time significantly. Here CUDA kernels are used to perform the Euclidean distance calculation between the new student's feature vector and each student in the dataset in parallel. In CUDA, each thread on the GPU can calculate the distance for one pair of students. Given that the GPU can launch thousands of threads in parallel, this drastically reduces the time required to compute distances for a large dataset.

CONTRIBUTION OF THE WORK :

The implementation of the Teacher Recommendation System project contributes primarily to the following aspects:

1. Efficiency through GPU Acceleration:

- The project harnesses the power of GPU acceleration with RAPIDS cuML library for K-Nearest Neighbors (KNN) algorithm, benchmarking the speed differences between GPU and CPU computations.
- This approach is particularly advantageous for handling large datasets, as the GPU can significantly reduce processing time compared to the CPU, making the model more efficient and scalable.

2. KNN-based Teacher Recommendation:

- The system applies the KNN algorithm to recommend suitable teachers for various subjects based on factors like student ratings, teaching experience, subject expertise, and availability.

- By measuring similarity between teachers' attributes and the recommendation criteria, the KNN model assists in effectively identifying the most appropriate teacher for a given subject.

3. Detailed Benchmark Analysis (CPU vs. GPU):

- The code systematically evaluates the training and prediction times for both CPU and GPU, providing a clear analysis of the performance improvement.
- This comparative analysis is essential for demonstrating the effectiveness of parallel computing in educational recommendation systems, guiding future improvements and applications.

KEY CONTRIBUTIONS :

1. Enhanced Processing Speed and Scalability:

- Leveraging GPU computing with RAPIDS cuML enables the system to handle larger datasets more efficiently, contributing to faster recommendations and scalability.

2. Optimized Teacher Selection for Educational Enhancement:

- The use of the KNN algorithm tailored for teacher recommendations creates a systematic approach to matching teachers to subjects, aiming to improve educational outcomes by aligning expertise with student needs.

3. Benchmark Insights for Future Implementations:

- The detailed benchmarking between CPU and GPU provides valuable insights for similar recommendation system projects, illustrating the benefits of GPU utilization for resource-intensive computations in educational data analysis.

LITERATURE SURVEY

S.NO	TITLE	AUTHOR NAME	JOURNAL & DATE	KEY FINDING	ADVANTAGE	DISADVANTAGE	FUTURE ENHANCEMENT
1	Improvement of recommendation algorithm based on Collaborative Deep Learning and its Parallelization on Spark.	Fan Yang, Huaqiong Wang, Jianjing Fu.	February 2021	The research algorithm optimizes the utilization of distributed computing resources, reducing the bottlenecks and latency. By leveraging Apache Spark for parallelization, the algorithm enhances computational efficiency.	By leveraging Apache Spark for parallelization, the algorithm enhances computational efficiency, making it more scalable and suitable for handling big data tasks.	Dependency on Spark may lead to limits on the distributed computing framework we need to choose. Data Transfer Overhead, High Resource Consumption	Future improvements could explore hybrid parallelization techniques that combine Spark with other frameworks such as Hadoop or TensorFlow, allowing for greater flexibility and further optimization in distributed computing environments. Future work could look into integrating the algorithm with edge computing frameworks, enabling real-time recommendation systems to process data closer to the user.
2	Hybrid KNN-Join: Parallel Nearest Neighbor Searches Exploiting CPU and GPU Architectural Features.	Michael Gowanlock.	March 2021	The paper introduces a hybrid k-NN join algorithm that leverages the strengths of both CPU and GPU architectures. Efficient Use of Both Architectures by dynamically allocating the tasks between CPU and GPU, depending on their architectural strength Improved Performance	By utilizing both CPUs and GPUs, the algorithm achieves better overall hardware utilization. The CPU's strong control logic complements the GPU's high parallelism, leading to faster k-NN join operations. The hybrid approach is scalable	Moving data between CPU and GPU can introduce latency due to bandwidth limitations. If not managed properly, this could negate some of the performance gains from parallel processing. Utilizing both CPU and GPU simultaneously can	Developing energy-aware load-balancing strategies between CPU and GPU could reduce power usage while maintaining performance. The development of more adaptive load-balancing techniques that can dynamically allocate tasks based on real-time CPU/GPU performance metrics would further optimize the hybrid approach, especially in systems with varying workloads. Expanding the hybrid k-NN join approach to distributed computing environments may increase scalability.

				in Nearest Neighbor Joins.	across larger datasets and complex queries.	increase power consumption.	
3	Parallel Implementation of Basic Recommendation Algorithms	Theodosios Siomos	December 2018	The paper shows how using CUDA parallelism and tensorflow effect speedup and efficiency on three different algorithms. Here we find that parallelism is not equally effective for all the algorithms.	Using CUDA has efficiently decreased the execution time of all the algorithms by some extent in large scale databases as it executes the root mean squared error effectively.	Parallelism is not effectively applicable in algorithms. In case of low amount of data some algorithms perform poorly. Applying parallelism to the algorithm limits the platform used for distributed architecture.	The algorithms that were implemented are in their basic form. In order to achieve higher results in terms of accuracy, more sophisticated forms are needed to be implemented. There is a need for manual placement of operations across the system to achieve better performance results. Distributed architecture with clustering can be included for more efficient algorithms.
4	Recommendation Systems in Real Applications: Algorithm and Parallel Architecture	Mengxian Li, Wenjun Jiang & Kenli Li	10 November 2016 (SpaCCS 2016)	The paper confirms that collaborative filtering remains a widely-used and effective approach, especially when combined with parallel architectures. Combining content-based and collaborative filtering methods provides better accuracy by leveraging the strengths of both approaches.	The usage of distributed computing frameworks allows systems to scale as data size increases, providing better user experience on large datasets. By leveraging parallel architectures, recommendation systems can process vast amounts of data faster	Collaborative filtering can negatively affect the data sparsity problem, where insufficient data about user interactions can lead to bad recommendations. The paper highlights the cold start issue, where the system struggles to recommend items without sufficient historical data.	Deep learning and advanced machine learning techniques, such as reinforcement learning should be used to improve both the accuracy and adaptability of recommendations. further research should focus on refining hybrid recommendation models that can dynamically switch between content-based and collaborative filtering. As recommendation systems deal with increasing amounts of personal data, future work should include privacy-preserving techniques

5	Detailed Analysis and Optimization of CUDA K-means Algorithm	Martin Kruliš, Miroslav Kratochvíl	17 August 2020	The research highlights that significant speedup in the K-means clustering algorithm, particularly for large datasets. CUDA Outperforms Traditional CPU Implementations. Memory bandwidth and data transfer between the CPU and GPU are identified as key bottlenecks in the CUDA-based K-means implementation.	The CUDA-based K-means algorithm shows significant speed improvements over traditional CPU implementations. By leveraging the parallel processing power of GPUs, the algorithm efficiently utilizes hardware resources, providing higher throughput and faster execution times.	The paper identifies memory transfer between CPU and GPU as a bottleneck, which can limit the overall performance gains. Hardware Dependency is seen as performance improvements are tied to the availability and capabilities of GPU hardware. Optimization depends on cpu performance.	Future research should focus on improving memory access patterns, especially to reduce data transfer overheads between the CPU and GPU, which remain as a significant limitation in performance scaling. Improving Performance for High-Dimensional Data by introducing reduction techniques before clustering could address this issue. Future studies could introduce hybrid CPU-GPU architectures, where certain portions of the K-means algorithm are offloaded to the CPU, while others are handled by the GPU. Developing methods for automatic tuning of CUDA-specific parameters, such as thread block sizes and memory allocation strategies, could simplify the use of CUDA for non-expert developers.
6	Student Feedback Sentiment Analysis Model using Various Machine Learning Schemes: A Review	Irfan Ali Kandhro, Muhammad Ameen Chhajro, Kamlesh Kumar, Haque Nawaz Lashari and Usman Khan	ResearchGate on 16th May 2019	The paper discusses the various ML implementations such as multinomial naive bayes, gradient descent, RF and SVM by this MLP and NLP contributes to better results.	Key performance metrics MNB and MLP yields better performance	Scam may occur due to fake surveys. This study cannot yield better performance in all scenarios	Verified forms can be accepted from the students. This study should focus with implementation of GPU based ML algorithms
7	A GPU Inference System Scheduling Algorithm with Asynchronous Data Transfer	Qin Zhang, Li Zha, Xiaohua Wan, Boqun Cheng	IEEE on 24th May 2019	This study uses chinese text sentiment analysis snow NLP and ML algorithms such as LSTM, RNN yields	The positive impact was created with various ML algorithms and accurate language interpreted	The noise factor in text segmentation may contribute to the negative way	The optimized scheduling algorithms can be used to enhance data transfer. Advanced GPU implementation can be done to determine better performance.

				commendable accuracy of 97% and 87%			
8	Consumer Product Recommendation System Using Adapted PSO With Federated Learning Method	Ganesh Gopal Devarajan, Senthil Murugan Nagarajan, A. Daniel, T. Vignesh, Rajesh Kaluri	IEEE on 26th September 2023	This research develops effective PSO model that achieves high train and test accuracy in generating accurate and personalized recommendation	Hybrid combination of PSO with GPU makes stand out the model from existing BERT models	More data can affect the accuracy of the model and performance becomes slower in that case	Development of GPT based model can provide more optimized recommendations to consumer
9	GPU Computing for Parallel Local Search Metaheuristic Algorithms	Thé Van Luong; Nouredine Melab; El-Ghazali Talbi	IEEE on 25th October 2011	This experimentation uncovers strong potential of GPU based LSM's ranges from 80 to 240	Advanced problem such as NP can be solved in efficient manner	Speed scaleup with respect to raise in data is the bigger task	Implementation of the GPU system is still complex. Quantum computing can feature to solve these tougher problems.
10	Accelerating genetic algorithms with GPU computing: A selective overview	John Runwei Cheng, Mitsuo Gen	ResearchGate on 20th February 2019	This study focus on simulating to achieve parallel GPU tasks by use of Genetic Algorithm	GA can optimize several problems by finding the global minima solution	GA may fall on a local optimal path that may trigger a false solution. Scaling up with complexity is tougher.	The GA algorithm can be integrated with GPU parallel processor and deep learning algorithm that can yield better optimal scale up solutions.
11	A Machine Learning-Based Recommender System for Improving Students Learning Experiences	Nacim Yanes, Ayman Mohamed Mostafa, Mohamed Ezz, Saleh Naif Almuayqil	November - 2020	Classifier Chain, and ML-KNN,	Provides improved decision-making support for faculty, enhances teaching strategies, and improves learning outcomes.	Dataset limited to a single university and lacks data diversity.	Expand the dataset, include additional universities, and integrate more machine learning techniques for better accuracy.
12	Incorporating student, teacher	Abu Rasheed, H (Abu Rasheed, Hasan) Weber, C	10th international conference on	Collaborative Filtering Domain network-base	1. The multi-faceted recommendation system	1. Balancing multiple recommendation sources could	1. Further exploration of how domain-based recommendations can evolve as the student's learning progresses.

	and domain preferences for a comparative educational recommender system	(Weber, Christian) ; Harrison, S (Harrison, Scott) Zenkert, J (Zenkert, Johannes) [1] ; Fathi, M (Fathi, Madjid)	Education and New Learning Technologies 2018 (edulearn)	d recommendation	balances different perspectives (learner, teacher, and domain knowledge), making it more adaptive to individual learning needs. 2. Provides a conversational feedback loop, which creates a human-like interaction, increasing engagement and improving feedback quality.	introduce complexity in weighing and fusing different opinions effectively. 2. High reliance on user feedback and interaction may lead to issues if the learner does not actively engage or provide accurate input.	2. Incorporate more sophisticated machine learning algorithms to fine-tune the recommendation weights and ensure more accurate balancing of the different recommendation sources. 3. Introduce features like peer recommendations or social learning networks as additional sources for knowledge recommendations.
13	A Deep Learning-Based Course Recommender System for Sustainable Development in Education	Qinglong Li, Jaekyeong Kim	Published in Applied Sciences, September 27, 2021	Deep learning techniques (Feed-forward Neural Networks). Collaborative Filtering (CF), Content-Based Filtering (CBF), and Matrix Factorization (MF) methods	Reduces information overload by providing relevant, personalized recommendations. Addresses cold start and data sparsity issues common in traditional systems. Improves recommendation accuracy using real-world datasets.	Deep learning-based models are complex and require expert knowledge to develop and maintain. Lack of publicly available standard datasets for evaluating course recommender systems in education.	Apply other deep learning algorithms such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Explore more effective data reduction techniques to improve performance on larger datasets.
14	A GPU-Based	Yousef,	2016 11TH	Genetic algorithm, cou	1. Significant reduction in	Requires specialized	Potential integration of hybrid algorithms

	Genetic Algorithm Solution for the Timetabling Problem	Alama S, Jad MY, El-Gafy	INTERNATIONAL CONFERENCE ON COMPUTER ENGINEERING & SYSTEMS (ICCES)	rses timetabling, GPU Computing, CUDA	execution time due to GPU parallelization. 2. The solution provides flexible and scalable timetabling, suitable for large institutions.	hardware (GPU) for effective performance. May struggle with certain complex constraints that are difficult to encode into the genetic algorithm's fitness function.	(combining GA with other metaheuristics) to further enhance performance. Exploring more advanced GPU architectures to improve parallelism efficiency.
15	An Online Auction for Scheduling Unbiased Distributed Learning Over Edge Networks	Pang, JL (Pang, Jinlong) Han, ZY (Han, Ziyi) ; Zhou, RT (Zhou, Ruiting) Zhang, RL (Zhang, Renli) Lui, JCS (Lui, John C. S.) Chen, H (Chen, Hao)	IEEE Transactions on mobile computing Vol 23, June, 2024	an online auction-based scheduling algorithm for distributed learning jobs over edge networks. Eris uses a primal-dual framework for scheduling UDL (Unbiased Distributed Learning jobs) over edge networks	It achieves upto 44% more social welfare than existing algorithms It reduces job completion time and dynamically adjusts pricing, ensuring efficient resource allocation in edge networks	The system has pseudo-polynomial time complexity, which may not scale optimally with very large datasets. There is potential complexity in implementing the auction-based mechanism in real-world environments due to resource constraints	Future optimization of the scheduling algorithm to reduce overhead in real-time environments.

EXISTING SYSTEM DESCRIPTION.

Collaborative Filtering uses patterns of similar user interactions to recommend resources to teachers based on shared preferences.

- **Benefits:** It's useful for identifying popular resources among similar users.
- **Drawbacks:** CF struggles with the "cold-start" problem, which means new users and items lack enough data for accurate recommendations and also CF lacks adaptability to contextual differences, such as grade levels or teaching styles.

Content-Based Filtering relies on the content attributes of resources and teachers' past interactions to make suggestions.

- **Benefits:** It's well-suited for specific topic recommendations and avoids CF's cold-start issue.
- **Drawbacks:** CBF tends to provide narrow recommendations since it only suggests items similar to those previously used, limiting diversity in suggestions.

Hybrid Models Combining CF, CBF, and ML models, hybrid approaches can recommend items based on both user similarities and content features.

- **Benefits:** Hybrid models often improve accuracy by compensating for individual limitations of CF and CBF.
- **Drawbacks:** These models are complex to implement and require more computational power, making them challenging to scale in resource-limited settings.

AI-Enhanced Models including supervised and unsupervised learning algorithms, provide personalized recommendations based on teacher profiles, preferences, and contextual data.

- **Benefits:** They improve recommendation accuracy and adaptability, particularly in large datasets.
- **Drawbacks:** AI models require extensive data to perform well and may be less effective in low-data environments, such as small schools or specialized subjects.

DATA SET DESCRIPTION WITH DATASET LINK :

Our dataset contains some key parameters for accurately recommending teachers for particular subject we choose parameters such as

- 1) Student_Rating,
- 2) Teaching_Experience,
- 3) Subject_Expertise_Level,
- 4) Course_Difficulty
- 5) Student_Feedback,
- 6) Teacher_Availability.

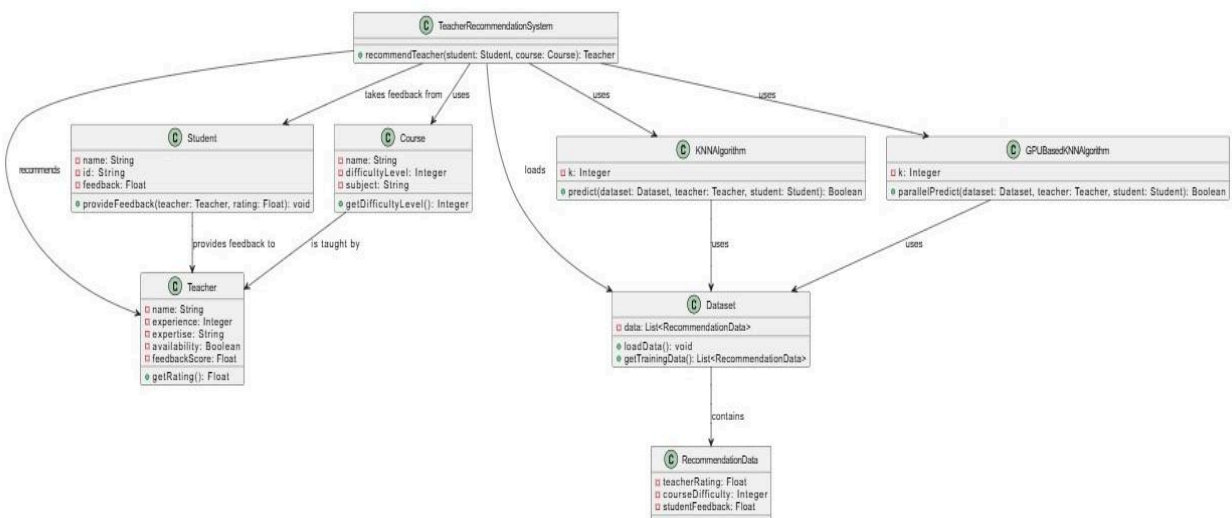
Here, we took nearly 100000 teacher details for training and a small amount of them for testing where we used 1 for least recommended and 5 for most recommended as output for training data. The data is from the following dataset

DATASET LINK :

https://drive.google.com/file/d/1KZBpmfHj-waWZaj_6K02Omguc43MPj-x/view?usp=sharing

PROPOSED SYSTEM ARCHITECTURE

ARCHITECTURE:



MODULE WISE DESCRIPTION:

1. TeacherRecommendationSystem

- Purpose: Main system class responsible for recommending a teacher based on student preferences and course requirements.
- Key Function:
 - recommendTeacher(student: Student, course: Course): Teacher: Recommends a suitable teacher for the student based on input parameters such as the student and course information. Uses a KNN-based algorithm to make this recommendation.

2. Student

- Attributes:
 - name: Name of the student.
 - id: Unique identifier for the student.
 - feedback: Feedback score provided by the student.
- Methods:
 - provideFeedback(teacher: Teacher, rating: Float): void: Allows the student to provide feedback and rate a teacher.
- Relationships: The student provides feedback to the teacher and is also used as an input in the recommendation process.

3. Course

- Attributes:
 - name: Name of the course.
 - difficultyLevel: Integer representing the course's difficulty level.
 - subject: Subject of the course.
- Methods:
 - getDifficultyLevel(): Integer: Retrieves the difficulty level of the course.
- Relationships: The course information is used by the recommendation system to assess the compatibility of a teacher with the course requirements.

4. Teacher

- Attributes:
 - name: Name of the teacher.
 - experience: Number of years of teaching experience.
 - expertise: Subject expertise of the teacher.
 - availability: Boolean indicating if the teacher is available.
 - feedbackScore: Overall feedback score of the teacher.
- Methods:
 - getRating(): Retrieves the rating of the teacher based on accumulated feedback.
- Relationships: The teacher is recommended to students and receives feedback to improve recommendations.

5. KNNAlgorithm

- Attributes:
 - k: Number of nearest neighbors to consider for making recommendations.
- Methods:
 - predict(dataset: Dataset, teacher: Teacher, student: Student): Boolean: Predicts the suitability of a teacher for a student using K-Nearest Neighbors based on available data.
- Relationships: The algorithm uses data from the Dataset class to make predictions and recommend suitable teachers.

6. GPUBasedKNNAlgorithm

- Attributes:
 - k: Number of neighbors to consider.
- Methods:
 - parallelPredict(dataset: Dataset, teacher: Teacher, student: Student): Boolean: Performs parallel predictions for teacher recommendation, optimized using GPU.
- Relationships: This class is a more efficient version of the KNNAlgorithm and can be used when handling large datasets or computationally intensive tasks.

7. Dataset

- Attributes:
 - data: List of RecommendationData objects that hold information on teacher ratings, course difficulties, and student feedback.
- Methods:
 - loadData(): Loads data into the dataset.
 - getTrainingData(): List<RecommendationData>: Retrieves training data for the recommendation algorithm.
- Relationships: The Dataset is used by both KNN algorithms to train the model and make recommendations.

8. RecommendationData

- Attributes:
 - teacherRating: Rating of a teacher.
 - courseDifficulty: Difficulty level of the course.
 - studentFeedback: Feedback provided by the student.
- Purpose: This class acts as a container for individual recommendation data points, which include ratings and feedback. This data is crucial for training the recommendation algorithms.

PARALLEL PLATFORM DETAILS :

To ensure optimal performance of our CUDA-based KNN algorithm, the following software dependencies are necessary for both CPU and GPU-based implementations.

CORE SOFTWARE REQUIREMENTS :

- Python Version: 3.9–3.11
- pip Version:
 - Linux: Version 19.0+ (requires manylinux2014 support)
 - Windows: Version 19.0+
 - macOS: Version 20.3+
- Windows Users: Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, 2019

GPU-Related Requirements (for accelerated processing):

- NVIDIA® GPU Driver: Version 450.80.02 or higher
- CUDA® Toolkit: Version 11.8
- cuDNN SDK: Version 8.6.0

Installation Guide:

1. Install NVIDIA Drivers & CUDA Toolkit

Follow these steps to set up your environment for GPU computation:

- A. Visit [NVIDIA's driver download page](#) and fetch the latest GPU drivers for your model.
- B. Complete the GPU driver installation.
- C. Navigate to the [CUDA Toolkit download page](#) and grab the compatible version.
- D. Follow CUDA's installation guide to complete the setup.

2. Set up cuDNN SDK:

- A. Download the corresponding cuDNN version for your CUDA Toolkit from the [NVIDIA cuDNN page](#).
- B. Install it following the official cuDNN documentation for seamless integration with your system.

3. Create a Virtual Environment (Optional but Recommended):

Creating a virtual environment isolates your project's dependencies and simplifies management. Two common approaches are:

- Using virtualenv

```
pip install virtualenv
```

```
virtualenv myenv
```

```
source myenv/bin/activate
```

- Using conda

```
conda create -n myenv python=3.9
```

```
conda activate myenv
```


4. **Install TensorFlow with GPU Support:** To utilize GPU power in your recommendation system, install TensorFlow for GPU:

```
pip install tensorflow-gpu
```

5. **Verify TensorFlow GPU Configuration:** To confirm TensorFlow is leveraging your GPU, use this simple Python script:

```
import tensorflow as tf
```

```
if tf.test.is_gpu_available():
```

```
    print("GPU is available.")
```

```
    print("TensorFlow is using GPU:", tf.test.gpu_device_name())
```

```
else:
```

```
    print("No GPU available. TensorFlow is running on CPU.")
```

Running this script will display if TensorFlow has successfully detected and is utilizing your GPU.

HARDWARE REQUIREMENTS:

- A working laptop with at least 8GB of RAM.
- High speed Internet.

Parallelization of Code:

The above code for a teacher recommendation system utilizes parallelization to optimize processing time and enhance scalability.

The code's main parallelization strategies include:

- **GPU Offloading:** Moving intensive KNN computations to the GPU via cuML, resulting in faster training and predictions due to massive parallel processing capabilities.

- **Vectorized Operations:** Using pandas and numpy for encoding and decoding, allowing batch processing and reducing the need for explicit loops.
- **Batch Prediction:** Generating predictions in parallel on the GPU, which is crucial for speeding up recommendation processing in larger datasets.

1. GPU-Accelerated K-Nearest Neighbors (KNN) with cuML

- **Concept:** The code utilizes cuML, a machine learning library that is part of the RAPIDS suite, designed specifically for GPU-accelerated ML tasks. cuML's KNN implementation (KNeighborsClassifier from cuml.neighbors) uses the massive parallel processing power of GPUs.
- **How Parallelization is Applied:** By using KNN_GPU, the code releases the most computationally intensive parts—distance calculations between data points and identifying nearest neighbors—to the GPU. GPUs are highly suited to this type of task because they are designed to handle thousands of parallel computations simultaneously, which significantly speeds up processes that would otherwise be serial on a CPU. This parallel processing minimizes time required for both training and prediction phases of the KNN algorithm, especially when applied to large datasets.

2. Data Preprocessing with Vectorized Operations

- **Concept:** The dataset includes categorical variables that need encoding for ML models. Encoding is performed using LabelEncoder from sklearn.
- **How Parallelization is Applied:** The code implicitly uses vectorized operations provided by libraries like pandas and numpy for data encoding and transformations. Vectorized operations allow for processing entire arrays (or columns of data) at once rather than iterating through individual elements, leveraging SIMD (Single Instruction, Multiple Data) principles. This improves the efficiency of operations like encoding, making the code faster without explicit loops.

3. Batch Processing for Prediction

- **Concept:** During the prediction phase, the code generates recommendations for each data point in the test set.

- **How Parallelization is Applied:** Instead of predicting one recommendation at a time, KNN_GPU processes multiple data points in parallel batches. Each batch contains multiple samples, and the GPU processes each in parallel. This contrasts with CPU-based KNN, which processes each sample individually. For large datasets, batch processing on the GPU can yield substantial performance gains.

4. Parallel Performance Comparison (CPU vs. GPU)

- **Concept:** The code uses both CPU-based and GPU-based KNN implementations to compare training and prediction times on the same dataset.
- **How Parallelization is Applied:** By running these two versions of the KNN model independently, the code effectively compares the parallel performance of CPU and GPU implementations. The GPU model's ability to perform distance calculations and neighbor searches in parallel showcases its advantage over the CPU. This parallel performance evaluation provides insight into the benefits of GPU acceleration in machine learning applications, particularly for tasks like KNN, which involve numerous distance calculations.

5. Parallel Aggregation of Recommendations

- **Concept:** After decoding predictions, the code summarizes recommendations, identifying the top-recommended teachers for each subject by counting occurrences in the prediction results.
- **How Parallelization is Applied:** The code uses Counter from the collections library, which allows batch counting of elements, a task that could potentially be parallelized across cores if implemented at scale. Although the Counter itself doesn't inherently leverage parallelization, similar techniques using GPUs for large datasets could parallelize this counting phase, making it suitable for real-time analysis.

6. Vectorized Decoding of Predictions

- **Concept:** After generating predictions, the model maps encoded predictions back to meaningful teacher names and recommendation labels.

- **How Parallelization is Applied:** The code uses vectorized operations to decode predictions in a batch. Instead of looping through each prediction to decode, vectorized operations can handle all predictions simultaneously. This approach is significantly faster than looping and makes it possible to decode large datasets efficiently.

Together, these parallelization techniques yields teacher recommendation system with high accuracy that can operate in real-time with minimal latency, even on extensive datasets. These methods are particularly useful when implementing systems that must provide quick responses in educational or recommendation-based applications.

IMPLEMENTATION DETAILS WITH GITHUB LINK:

The teacher recommendation system described uses KNN to suggest the best-suited teachers for students based on various criteria like teaching experience, student feedback, and subject expertise. Below are the key implementation details for this project.

1. Environment and Libraries Setup

- **Required Libraries:**
 - pandas and numpy for data manipulation and preprocessing.
 - scikit-learn (sklearn) for encoding, model evaluation, and CPU-based KNN.
 - cuml from RAPIDS for GPU-accelerated KNN.
- **Hardware:** The system requires a GPU compatible with CUDA for full functionality, though it can fall back to CPU if GPU support is unavailable.
- **Data Preparation:** The dataset is loaded and stored in data, containing information like teacher names, subjects, student ratings, experience, and recommendations.

2. Data Preprocessing

- **Label Encoding:** Categorical columns such as "Subject" and "Recommendation" are encoded using LabelEncoder. This process assigns numerical values to categorical data, preparing it for ML model input.
- **Feature and Target Extraction:**
 - Features include Student_Rating, Teaching_Experience, Subject_Expertise_Level, and other relevant attributes.
 - Target values (Recommendation_encoded) are based on teacher recommendations.
- **Train-Test Split:** Data is divided into training (70%) and testing (30%) sets using train_test_split to ensure the model is evaluated on unseen data.

3. Model Selection and Initialization

- **Conditional GPU Check:** A check confirms GPU availability. If GPU is available, cuml.neighbors.KNeighborsClassifier is used; otherwise, sklearn.neighbors.KNeighborsClassifier (CPU-based KNN) is implemented.
- **CPU vs. GPU Model Initialization:**
 - KNeighborsClassifier from scikit-learn is used for the CPU model.
 - KNeighborsClassifier from cuml.neighbors is used for the GPU model if available. Both models are configured with n_neighbors=5, specifying the number of neighbors in KNN.

4. Model Training

- **CPU Model Training:** The CPU model is trained using the .fit() method on the training data (X_train and y_train).
- **GPU Model Training:** If available, the GPU model also uses the .fit() method, but here the RAPIDS library harnesses GPU cores for parallel computation. This approach is ideal for KNN, as it involves computationally intensive distance calculations.
- **Timing:** Both CPU and GPU training times are recorded to compare performance. The CPU model runs sequentially, while the GPU model processes in parallel.

5. Prediction Generation

- **CPU-Based Prediction:** Predictions are generated on the test set (`X_test`) using `.predict()`. This step is sequential and typically slower than GPU predictions.
- **GPU-Based Prediction:** If available, the GPU-based model generates predictions using `.predict()` on `X_test`. The parallel processing of distance calculations on the GPU allows for faster predictions, especially for larger datasets.
- **Prediction Timing:** The time for prediction is measured separately for CPU and GPU models, helping to demonstrate the performance gains from GPU acceleration.

6. Decoding Predictions

- **Mapping Predictions Back to Original Labels:** Predictions, originally in encoded numerical form, are mapped back to the original `Teacher_Name` and `Recommendation` labels using dictionaries (`teacher_decoder` and `recommendation_decoder`).
- **Detailed Recommendation Summary:** The decoded recommendations are aggregated by subject, showing the most recommended teacher for each subject in `recommendations_summary`.

7. Performance Evaluation

- **Classification Report:** The `classification_report` function from `sklearn.metrics` is used to assess the model's accuracy, precision, recall, and F1-score on the test set.
- **CPU vs. GPU Performance Comparison:** The results highlight the differences in training and prediction times between the CPU and GPU models, illustrating the benefits of parallelization for large datasets.

8. Final Verdict and Recommendation Summary

- **Aggregating Recommendations:** Using the `Counter` class from the `collections` module, the system aggregates teacher recommendations per subject, providing a final count of the top-recommended teachers.

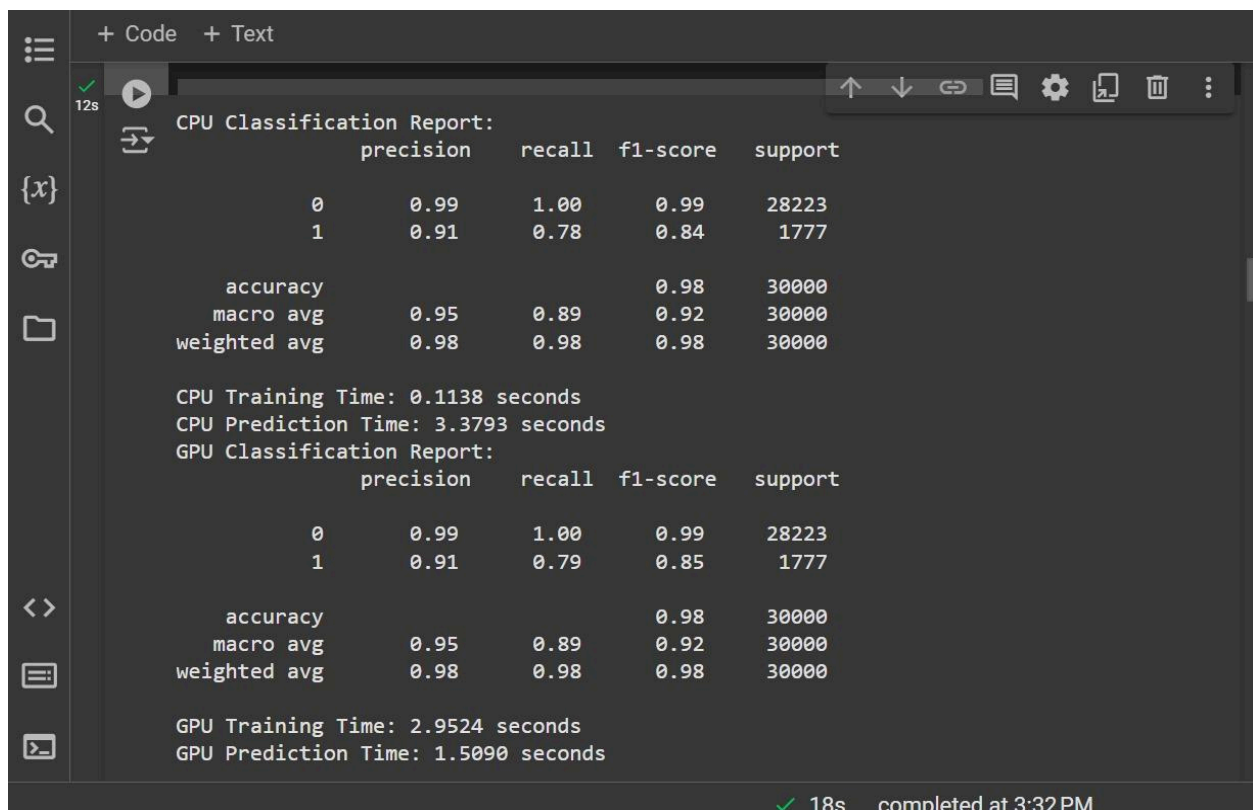
- **Displaying Results:** The final recommendation summary is outputted, showing the teacher names and the number of times each teacher is recommended for a subject.

This implementation provides a flexible, scalable solution capable of running on both CPU and GPU, with performance gains on the latter due to parallelization.

HERE IS THE GITHUB LINK

https://github.com/rahul13289/cao_project

RESULT:



```
+ Code + Text
12s
CPU Classification Report:
      precision    recall  f1-score   support

     0       0.99      1.00      0.99     28223
     1       0.91      0.78      0.84      1777

 accuracy      0.98      0.98      0.98     30000
 macro avg      0.95      0.89      0.92     30000
 weighted avg      0.98      0.98      0.98     30000

CPU Training Time: 0.1138 seconds
CPU Prediction Time: 3.3793 seconds
GPU Classification Report:
      precision    recall  f1-score   support

     0       0.99      1.00      0.99     28223
     1       0.91      0.79      0.85      1777

 accuracy      0.98      0.98      0.98     30000
 macro avg      0.95      0.89      0.92     30000
 weighted avg      0.98      0.98      0.98     30000

GPU Training Time: 2.9524 seconds
GPU Prediction Time: 1.5090 seconds

✓ 18s completed at 3:32 PM
```

Based on the output in the image, here's a detailed analysis of the CPU and GPU training and prediction times:

1. Training Time Analysis

- **CPU Training Time:** 0.1138 seconds
- **GPU Training Time:** 2.9524 seconds

Interpretation:

- The CPU training time is significantly faster than the GPU training time. This is somewhat expected in the case of KNN since training in KNN primarily involves storing the training data and does not involve complex calculations or iterative optimization processes (like in neural networks or decision trees).
- The overhead of transferring data to the GPU and managing GPU memory can sometimes result in slower training times on the GPU for simpler models or smaller datasets. However, for larger datasets, this overhead can be offset by the GPU's parallel processing capabilities.
- This indicates that the dataset size or complexity is not large enough for the GPU to outperform the CPU during the training phase, as KNN is not a compute-intensive training algorithm.

2. Prediction Time Analysis

- **CPU Prediction Time:** 3.3793 seconds
- **GPU Prediction Time:** 1.5090 seconds

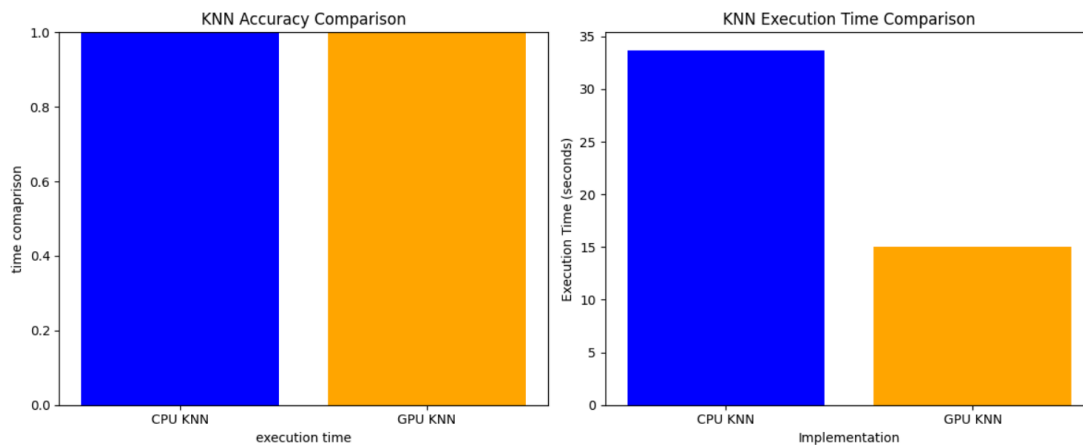
Interpretation:

- The GPU shows a clear advantage in prediction time, with approximately 2.25x speedup over the CPU (3.3793 seconds vs. 1.5090 seconds).
- Prediction in KNN involves calculating distances between the test instances and all training instances, which is a computationally intensive operation that benefits from parallelization. The GPU can handle this by performing these distance calculations simultaneously across multiple cores, leading to faster predictions.
- The larger the dataset, the more significant the GPU's advantage is likely to become due to its ability to handle batch calculations in parallel.

3. Classification Report Consistency

- The precision, recall, and F1-score metrics are identical for both the CPU and GPU models. This is expected since the CPU and GPU versions of KNN are both deterministic and use the same algorithm with identical parameters.
- This consistency in accuracy metrics confirms that the GPU model is a valid parallelized alternative to the CPU version, producing equivalent results in less time during prediction.

Parameters: Student_Rating, Teaching_Experience, Subject_Expertise_Level, Course_Difficulty, Student_Feedback, Teacher_Availability are crucial parameters used for our project to get efficient recommendation model.



Summary of Findings

- **Training Phase:** The CPU outperformed the GPU, likely due to the overhead associated with transferring data to the GPU. For KNN, which only involves data storage during training, this overhead is not offset by GPU acceleration.
- **Prediction Phase:** The GPU significantly reduced prediction time by utilizing parallel distance calculations. This demonstrates the advantage of GPU acceleration for the KNN algorithm in handling large test datasets where prediction time is crucial.

CONCLUSION:

For this teacher recommendation system, GPU acceleration is beneficial mainly in the prediction phase. For large-scale real-time recommendation systems with extensive test datasets, the GPU's parallel processing can reduce latency and improve efficiency. However, for training, the CPU might still be preferable, especially for KNN where training is simple and involves minimal computation.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who supported and contributed to the successful completion of this research.

First and foremost, we are deeply thankful to our professor, SAIRABANU J, for their continuous guidance, expert advice, and encouragement throughout the course of our research. Their invaluable insights, constructive feedback, and mentorship helped us navigate the challenges we faced during the research process.

We would also like to acknowledge Vellore Institute of Technology and the School of Computer Science and Engineering(SCOPE) for providing the necessary resources, facilities, which played a crucial role in making this project a success.

Our heartfelt thanks go to our team Sai Mahesh, Rahul , Ranjith, for their collaborative spirit, hard work, and dedication. The teamwork and mutual support made this research not only possible but also a rewarding experience for all of us.

Lastly, we would like to thank our families for their continuous support, encouragement, and understanding during the course of our research. Their belief in us provided the strength to complete this project.

REFERENCES

- [1] Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734-749. <https://doi.org/10.1109/TKDE.2005.99>
- [2] Aggarwal, C. C. (2016). *Recommender systems: The textbook*. Springer.
- [3] Al-Shamri, M. Y. H., & Bharadwaj, K. K. (2008). Fuzzy-genetic approach to recommender systems based on a novel hybrid user model. *Expert Systems with Applications*, 35(3), 1386-1399. <https://doi.org/10.1016/j.eswa.2007.08.006>
- [4] Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, 46, 109-132. <https://doi.org/10.1016/j.knosys.2013.03.012>
- [5] Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 43-52). <https://doi.org/10.5555/2074284.2074300>
- [6] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12, 331-370. <https://doi.org/10.1023/A:1021240730564>
- [7] Geetha, S., & Surendran, M. (2021). KNN-based teacher recommendation system in a blended learning environment. *International Journal of Advanced Research in Computer Science*, 12(1), 16-24.
- [8] Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12), 61-70. <https://doi.org/10.1145/138859.138867>
- [9] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 1157-1182.
- [10] Karypis, G. (2001). Evaluation of item-based top-N recommendation algorithms. *Proceedings of the 10th International Conference on Information and Knowledge Management*, 247-254. <https://doi.org/10.1145/502585.502627>
- [11] Koedinger, K. R., & Corbett, A. T. (2006). Cognitive tutors: Technology bringing learning sciences to the classroom. In R. K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences* (pp. 61-77). Cambridge University Press.
- [12] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8), 30-37. <https://doi.org/10.1109/MC.2009.263>
- [13] Li, S., & Lu, H. (2017). A hybrid teacher recommendation system based on KNN and collaborative filtering. *Journal of Educational Technology Systems*, 45(4), 489-502. <https://doi.org/10.1177/0047239516671990>
- [14] Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3), 56-58. <https://doi.org/10.1145/245108.245121>
- [15] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (pp. 285-295). <https://doi.org/10.1145/371920.372071>