# Database Management Systems Course

## Name: Harun Yahya Ünal

Project description: This Project is an AI based tutoring solution for the Database Management Course. The AI is used for the following features:

* Answering students’ questions about the course
* Measuring the student's level of knowledge about the course

### Other non-AI based features implemented include the following.

* In the quiz section, it gives score to the student based on the number of correct answers. This score is used to calculate the overall score.

### Features that may be implemented in the future are as follows

* A model may be implemented to generate feedback according to the overall score.
* A study part may be implemented to provide course notes for the section that has low overall score.
* A practice part may be implemented to practice that student can write a database code then ai evaluates the code, gives feedback.

### The Project used the following AI models

|  |  |  |
| --- | --- | --- |
| Model Name | Reference (downloaded from) | Used for |
| distilbert-base-cased-distilled-squad | Hugging Face Transformers | It is used for answering the questions of students about a selected section. |
| all-MiniLM-L6-v2 | Hugging Face Sentence Transformers | It is used for generating a knowledge score for student for a selected section. |

### Describe your data model and how it is implemented here

1. **DistilBERT-Based Question Answering Model**

It is a pre-trained Transformer-based NLP model (DistilBERT fine-tuned on the SQuAD dataset).

The model finds the most relevant answer based on a question and a context. It takes the question and context as input, turns them into embeddings, processes them with the DistilBERT model, and identifies the start and end positions of the answer.

Takes two strings as question and context.

Give a dictionary with the answer, start index, and confidence score.

The Hugging Face transformers library is used to load the model (distilbert-base-cased-distilled-squad) and pipeline. The --pipeline("question-answering")-- simplifies the process by handling both preprocessing and postprocessing, making it easier to use. When a question and context are given to the model, it breaks the text into tokens, processes them using the Transformer architecture, and finds the answer span based on token probabilities.

1. **Text Similarity Model with SentenceTransformer**

It is embedding-based data representation using all-MiniLM-L6-v2.

The model converts textual data (sentences or documents) into dense vector embeddings in a high-dimensional space. The embeddings are optimized for semantic similarity calculations.

Takes one string as student input and the list of rules.

Gives dense embeddings (vectors) for both the student input and rules. And cosine similarity scores between the embeddings to measure their semantic closeness.

The SentenceTransformer model (all-MiniLM-L6-v2) from the sentence-transformers library is used to turn text into vector forms. Cosine similarity (util.cos\_sim) is used to compare the embeddings of the student input and the rules to check how well the student's answer matches the reference rules.

The similarity scores are further adjusted based on: Length of the student input (penalties for overly short or excessively long responses), average similarity to all rules, scaled to a percentage.

### Describe your modules: what it does, how it does in here including the source code for each module

1. **DistilBERT-Based Question Answering Model**

This module extracts the most relevant answer from a given context based on a provided question. It simplifies Question Answering tasks using a pre-trained DistilBERT model.

The model uses the Hugging Face pipeline API for easy access to the pre-trained DistilBERT model fine-tuned on the SQuAD dataset. It takes two inputs: a question and a context then tokenizes the inputs internally (handled by pipeline), processes them through the Transformer model, and predicts the answer span based on token probabilities.

Source Code:  
from transformers import pipeline

from sentence\_transformers import SentenceTransformer, util

# Question Answering Model is loading.

qa\_model = pipeline("question-answering", model="distilbert-base-cased-distilled-squad")

def generateAnswer(question,section\_info):

    result = qa\_model(question=question, context=section\_info)

    answer = result['answer']

    return answer

1. **Text Similarity Model with SentenceTransformer**

This module evaluates the similarity between a student's input and predefined rules. It scores how closely the student's response matches the rules using sentence embeddings and cosine similarity.

The model loads predefined rules from a JSON file, converts both the student's input and the rules into dense vector representations using the SentenceTransformer model then computes cosine similarity between the vectors to find the most similar rule(s). Finaly, adjusts the score based on the length of the student's input and excessive verbosity.

Source Code:

import json

from sentence\_transformers import SentenceTransformer, util

def load\_rules(input\_file="rules.json"):

    """Load rules from JSON"""

    try:

        with open(input\_file, "r") as file:

            rules = json.load(file)

        return rules

    except FileNotFoundError:

        print("Rules file was not found.")

        return {}

def evaluate\_student\_input(sectionId, student\_input, rules\_file="rules.json"):

    # 1. Load the rules

    rules = load\_rules(rules\_file)

    if sectionId not in rules:

        print(f"Could not found rules for '{sectionId}'")

        return 0.0

    section\_rules = rules[f"{sectionId}"]

    # 2. Sentence Transformers model

    similarity\_model = SentenceTransformer('all-MiniLM-L6-v2')

    # 3. Embedding the student\_input and rules

    student\_embedding = similarity\_model.encode(student\_input, convert\_to\_tensor=True)

    rule\_embeddings = similarity\_model.encode(section\_rules, convert\_to\_tensor=True)

    # 4. Calculating the similarity score.

    scores = util.cos\_sim(student\_embedding, rule\_embeddings).max(dim=1)[0]

    # 5. Average of the scores

    average\_score = scores.mean().item() \* 100

    # 6. Add penalty based on length of student\_input

    # Penalty for short inputs

    if len(student\_input.split()) < 10:  # Limits for short inputs

        average\_score \*= 0.5  # Halve the score

    # 7. Add penalty for too much information

    # If text length exceeds rules, apply penalty

    rule\_terms = sum([len(rule.split()) for rule in section\_rules])

    student\_terms = len(student\_input.split())

    if student\_terms > rule\_terms \* 1.5:  # If the student text is 1.5 times longer

        average\_score -= 10  # Lower the score by 10 points

    # Limiting the score to not exceed 100%

    average\_score = min(average\_score, 100)

    return average\_score

## Main function

#if \_\_name\_\_ == "\_\_main\_\_":

#    student\_input = "SQL is used for querying and manipulating data,there are four operations, they are select,delete,insert,update. delete is used for removing data,"

#    sectionId = "4"

#    score = evaluate\_student\_input(sectionId, student\_input)

#    print(f"'Değerlendirme sonucu: {score:.2f}%")

## Main function

#if \_\_name\_\_ == "\_\_main\_\_":

#    student\_input = "SQL is used for querying and manipulating data,there are four operations, they are select,delete,insert,update. SELECT is used for retrieving data, delete is used for removing data,"

#    sectionId = "4"

#    score = evaluate\_student\_input(sectionId, student\_input)

#    print(f"'Değerlendirme sonucu: {score:.2f}%")

rules.json:

{

    "2": [

        "A Database Management System (DBMS) is software designed to store, retrieve, manage, and manipulate data in a structured manner",

        "It replaces traditional file systems by addressing issues such as data redundancy, inconsistency, and lack of scalability",

        "A DBMS provides several key features, including data abstraction, data independence, and support for multiple users",

        "It consists of components like the database engine, which processes queries, and the storage manager, which handles the organization of data on physical storage",

        "Real-world applications include inventory management systems, customer relationship management (CRM) tools, and banking systems",

        "By understanding the role and components of a DBMS, one can appreciate its importance in managing large-scale data efficiently"

    ],

    "3": [

        "The relational model represents data in the form of tables, known as relations, which consist of rows (tuples) and columns (attributes)",

        "Each table has a primary key that uniquely identifies its rows, and foreign keys establish relationships between different tables",

        "This model ensures data consistency and eliminates redundancy by organizing information into logically related tables",

        "Operations such as selection, projection, and join allow for efficient data retrieval and manipulation",

        "The relational model also enforces constraints like entity integrity, which ensures no null values in primary keys, and referential integrity, \n    which maintains valid relationships between tables",

        "Understanding the relational model is fundamental to designing structured and scalable databases"

    ],

    "4": [

        "Structured Query Language (SQL) is the standard language used to interact with relational databases",

        "It provides commands for data definition, manipulation, and querying",

        "For example, the CREATE TABLE statement defines the structure of a database table, while INSERT, UPDATE, and DELETE allow for modifying data",

        "The SELECT statement is used to query data, with additional clauses like WHERE for filtering, GROUP BY for aggregation, and ORDER BY for sorting",

        "SQL also supports joins, enabling the combination of data from multiple tables",

        "Mastery of SQL is essential for working with relational databases, as it allows users to extract meaningful insights and manage data effectively"

    ],

    "5": [

        "Normalization is the process of organizing database tables to reduce redundancy and improve data integrity",

        "It involves dividing a database into smaller, related tables and defining relationships between them",

        "The process is guided by normal forms, such as the First Normal Form (1NF), which eliminates duplicate columns; \n    the Second Normal Form (2NF), which removes partial dependencies; and the Third Normal Form (3NF), which eliminates transitive dependencies",

        "By normalizing a database, one can ensure that it is free from update, delete, and insert anomalies",

        "Although normalization improves efficiency and consistency, it may sometimes require denormalization for performance optimization in certain applications"

    ],

    "6": [

        "Transactions are sequences of database operations treated as a single unit of work",

        "They adhere to the ACID properties: Atomicity ensures all operations are completed or none are applied; \n    Consistency guarantees the database remains in a valid state; Isolation prevents concurrent transactions from interfering with each other; \n    and Durability ensures changes are permanent even in case of a failure",

        "Concurrency control mechanisms, such as locking and timestamps, are used to manage simultaneous access to the database",

        "Common issues include dirty reads, where a transaction reads uncommitted data, and deadlocks, \n    where two transactions wait indefinitely for resources held by each other",

        "Effective transaction management ensures data integrity and system reliability in multi-user environments"

    ]

}

### Describe how your program runs including the libraries required for it to run

For the websites, the program uses ASP.NET with MVS architecture to display the websites. User requests (which page the user wanted to display, some operations such as GetOverallScore, checkQuizAnswer, UpdateStudentScore) are send with HTTP protocols to the controller of each page. Then the controller sends back a response for each request in json format. We use controllers to navigate the user between pages too with action methods. To access the database, entity framework was used. We have a repository script for each table in the database. We can access the data with this repository scripts easily.

For the communication between the AI part of the program, RESTful API is used with Flask library. It is required installing flask library with “pip install flask” to use API gateway. ASP.NET sends the data to the python program with RESTful API, then the python program gets the data with “post” method. Two model uses same way to gets the data from ASP.NET but the uses different api address. For the Question-Answering model “/api/answer” is used to send the answer. For the text Similarity model “/api/similarityScore” is used to send the score.

The AI part of the program is written in Python, and it works with LLM models. Two types of models are used: Question-Answering and Text Similarity.

For the Question-Answering model, the program uses the Hugging Face transformers library. It uses a pre-trained model called distilbert-base-cased-distilled-squad to find answers for given questions and context (section information). To use this, you need to install the transformers library with the command: pip install transformers.

For the Text Similarity model, the program uses the Hugging Face sentence-transformers library. It uses the all-MiniLM-L6-v2 model to measure cosine similarity. To use this model, you need to install the sentence-transformers library with the command: pip install sentence-transformers. This library also needs another library, torch, for tensor operations.

The APIs receive input and return output in JSON format. This format makes it easy to integrate the program with other systems or applications.

The program also uses some predefined rules to check student inputs. Without these rules, the similarity model might give wrong scores, especially if the input is too short. To fix this, the program adds penalties for very short or very long inputs. This way, the scores are more accurate and useful.

**API Examples:**

* **For Question-Answering:**

Input:

{

"question": "What is Python?",

"sectionInfo": "Python is a programming language that lets you work quickly."

}

Output:

{

"success": true,

"answer": "a programming language"

}

* **For Text Similarity:**

Input:

{

"studentInput": "Python is a fast and easy programming language.",

"sectionId": "2"

}

Output:

{

"success": true,

"similarityScore": 85.75

}

### Write a user manual how a user uses your program