For your use case—**generating course recommendations** based on job descriptions, transcript details, and available course lists—an ideal approach would likely involve combining several models and components to handle different parts of the task effectively. Here’s a high-level **system design** and flow for your use case:

**1. Data Preprocessing and Representation**

* **Input**: Job descriptions, student transcripts (course history), available courses with descriptions.
* **Preprocessing**:
  + **Job Description**: Extract key skills, responsibilities, and required competencies. Use **Named Entity Recognition (NER)** or **text classification** to extract relevant skills from the job description.
  + **Student Transcript**: Extract current courses, grades, and past course performance. This can be processed into a structured format (e.g., a list of course names, grades, credits).
  + **Available Courses**: Normalize and structure course data, linking each course to its description, skillset, prerequisites, etc.
* **Goal**: Transform these inputs into structured data that the models can use (e.g., skills extracted from job description, courses taken, etc.).

**2. Skill Matching and Recommendation Model**

* **Job Description Analysis**:
  + Use a **BERT-based model** (such as **BERT for NER**) or **T5** for extracting the key **skills, responsibilities**, and **competencies** mentioned in the job description.
  + The model could also generate a **summary of required skills** for the job description, which could be used to identify what skills the student needs to acquire.
* **Student’s Skillset**:
  + Use a **text embedding model** (e.g., **Sentence-BERT** or **T5**) to process the transcript, converting it into embeddings representing the skills already acquired by the student (based on course history).
  + **Skills Gap Analysis**: Compare the skills required by the job (from the job description) with the student's existing skillset (from transcript), identifying the **gap**.
* **Available Course Match**:
  + Use a **similarity-based model** (e.g., **BERT embeddings** or **Sentence-BERT**) to match the extracted **skills** from the job description and **student's existing skills** to courses available in the catalog.
  + **Rank Courses**: Generate a ranked list of courses based on how well they align with the student's skill gaps.

**3. Course Recommendation Generation**

* Once you have the skill gaps and ranked course matches, the next step is to **generate recommendations**.
* **Text Generation Model**:
  + Use a **GPT-2** or **T5** model to generate natural language **recommendations** based on the job description, student transcript, and available courses. This model will take into account the required skills and suggest courses that will help the student fill the skill gaps.
  + The output should include the **specific courses** the student should take, with a brief description explaining why each course is a good match.
* **Recommendation Filter**:
  + Consider constraints like credit hours (e.g., minimum of 8 credits, maximum of 18 credits).
  + Include a filtering step to ensure that the recommended courses respect the student’s past course history and prerequisites.

**4. Architecture Flow**

Here's the **system design flow**:

**Step 1: Input Processing**

* **Input**: Job description, student transcript, available courses
* Extract **key skills and competencies** from the job description (NER, skill extraction).
* Convert **transcript details** into a skill-based representation (e.g., course names, grades, credits).

**Step 2: Skill Gap Analysis**

* Use a **text embedding model** (like **Sentence-BERT**) to convert both job description and student transcript into vectors.
* Compare the **job's required skills** (from the job description) with the **skills the student has already acquired** (from transcript data).
* Identify **skills gaps** (skills the student needs to acquire).

**Step 3: Course Matching**

* **Course Matching**: Use a **semantic search** (via embeddings) to match available courses to the identified **skills gaps**.
* Rank the available courses by how well they help the student fill the skills gap.

**Step 4: Course Recommendation Generation**

* Use a **text generation model** (like **GPT-2** or **T5**) to generate human-readable course recommendations, explaining why each course is relevant for bridging the skill gaps.
* Ensure recommendations follow **credit constraints** (8–18 credits).

**Step 5: Output**

* **Final Output**: A list of **recommended courses**, with a description of each course, why it’s recommended, and how it aligns with the job requirements and the student’s skill gaps.

**5. Additional Considerations**

* **Scalability**: Ensure that the system can handle multiple students and large course catalogs. Consider using batch processing for efficiency.
* **Feedback Loop**: Optionally, the system can include a **feedback loop** where the student’s performance (e.g., grades in recommended courses) helps refine future recommendations.

**Architecture Components:**

1. **Data Collection**: Pre-process input data (job descriptions, transcripts, course lists).
2. **Skill Extraction Model**: Use **BERT**, **T5**, or **GPT** to extract and match skills.
3. **Embedding Model**: Use **Sentence-BERT** for similarity matching.
4. **Text Generation Model**: Use **GPT-2** or **T5** for generating personalized course recommendations.
5. **Recommendation Engine**: Filter and rank courses based on skills and constraints (credits, prerequisites).
6. **User Interface**: Output the recommendations in a user-friendly format, possibly through a web or mobile app.

**6. End-to-End Flow Diagram:**

* **Inputs**: Job Description + Transcript Data + Course Catalog
* **Processing**: Skill Extraction → Skill Gap Analysis → Course Matching
* **Model(s) Used**: BERT (NER, Skill Extraction), Sentence-BERT (Similarity Matching), GPT-2/T5 (Text Generation)
* **Outputs**: Personalized Course Recommendations (with reasoning and constraints)

**Conclusion:**

* **Primary Models**: **BERT** (for extracting skills and competencies) and **GPT-2** or **T5** (for generating course recommendations).
* **Supporting Models**: **Sentence-BERT** (for embedding-based similarity and matching).
* This system leverages multiple models in a pipeline to provide personalized, skill-based course recommendations based on job descriptions, student transcripts, and available course data.

Let me know if you need more details or help with a specific component of the design!

Models according to input type.

**1. Models with Longer Output Capabilities**

These models support longer max\_length during generation. The following can generate up to or beyond 2048 tokens:

**GPT-NeoX-based Models (EleutherAI)**

• [**GPT-NeoX-20B**](https://huggingface.co/EleutherAI/gpt-neox-20b)

• Supports very large outputs (up to ~4096 tokens).

• Highly efficient and suitable for tasks requiring longer responses.

**MosaicML MPT Series**

• [**MPT-7B**](https://huggingface.co/mosaicml/mpt-7b)

• Can handle up to **65,000 tokens** in certain configurations (MPT-7B-StoryWriter).

• Best for long-form generation.

**LLaMA 2-based Models**

• [**LLaMA 2-13B**](https://huggingface.co/meta-llama/Llama-2-13b-chat)

• Typically supports generation up to **4096 tokens** in standard configurations.

• Variants like LLaMA 2-7B-32K (TogetherComputer) can go beyond **32,000 tokens** for both input and output.

**XGen-7B**

• [**XGen-7B-8K**](https://huggingface.co/Salesforce/xgen-7b-8k-base)

• Fine-tuned for efficiency and longer outputs.

• Can handle up to **8000 tokens** in total length (input + output).