

Enhanced Questions API with Attribute Vectors

Overview

The enhanced questions endpoint returns questions from any hierarchical level with:

- **All attributes available at that level** (consolidated across all topics)
 - **Q-matrix in binary vector form** for each question
 - **Efficient attribute mapping** for machine learning and analysis
-

Endpoint

```
GET /api/hierarchy/{level}/{item_id}/questions/enhanced
```

Parameters

Path Parameters:

- **level** (required) - Hierarchical level: **exam**, **class**, **subject**, **chapter**, or **topic**
- **item_id** (required) - UUID of the item at that level

Query Parameters:

- **page** (optional) - Page number (default: 1)
 - **page_size** (optional) - Items per page (default: 20)
-

Response Structure

```
{
  "level": "chapter",
  "level_id": "chapter-uuid",
  "total_questions": 50,
  "attribute_count": 12,
  "attributes": [
    {
      "id": "attr-uuid-1",
      "name": "Problem Solving",
      "description": "Ability to solve complex problems",
      "topic_id": "topic-uuid-1"
    },
    {
      "id": "attr-uuid-2",
      "name": "Formula Application",
      "description": "Applying mathematical formulas",
      "topic_id": "topic-uuid-1"
    }
  ]
}
```

```

    },
    ...
  ],
  "questions": [
    {
      "id": "question-uuid-1",
      "content": "What is the value of x?",
      "options": ["1", "2", "3", "4"],
      "correct_answer": "2",
      "difficulty": 0.5,
      "discrimination": 1.2,
      "guessing": 0.25,
      "topic_id": "topic-uuid-1",
      "q_vector": [1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
      "attribute_count": 4
    },
    ...
  ],
  "pagination": {
    "total": 50,
    "page": 1,
    "page_size": 20,
    "total_pages": 3,
    "has_more": true
  }
}

```

Key Features

1. Unified Attribute List

All attributes from all topics at the specified hierarchical level are consolidated into a single list with consistent indexing.

2. Binary Q-Vector

Each question has a **q_vector** - a binary array where:

- **Index** corresponds to the position in the **attributes** array
- **Value** is **1** if the attribute applies to this question, **0** if not

3. Attribute Count

Each question includes **attribute_count** - the sum of its **q_vector** (number of attributes that apply).

Example Usage

Get Questions from a Chapter

```
curl "http://localhost:5200/api/hierarchy/chapter/123e4567-e89b-12d3-a456-426614174000/questions/enhanced?page=1&page_size=10"
```

Response Example

```
{
  "level": "chapter",
  "level_id": "123e4567-e89b-12d3-a456-426614174000",
  "total_questions": 25,
  "attribute_count": 8,
  "attributes": [
    {
      "id": "attr-1",
      "name": "Algebraic Manipulation",
      "description": "Simplifying algebraic expressions",
      "topic_id": "topic-1"
    },
    {
      "id": "attr-2",
      "name": "Equation Solving",
      "description": "Solving linear equations",
      "topic_id": "topic-1"
    },
    {
      "id": "attr-3",
      "name": "Graphing Skills",
      "description": "Plotting graphs",
      "topic_id": "topic-2"
    },
    {
      "id": "attr-4",
      "name": "Interpretation",
      "description": "Interpreting graph data",
      "topic_id": "topic-2"
    }
  ],
  "questions": [
    {
      "id": "q1",
      "content": "Solve:  $2x + 5 = 13$ ",
      "options": ["4", "6", "8", "10"],
      "correct_answer": "4",
      "topic_id": "topic-1",
      "q_vector": [1, 1, 0, 0, 0, 0, 0, 0],
      "attribute_count": 2
    },
    {
      "id": "q2",
      "content": "Plot  $y = 2x + 1$ ",
      "options": [...],

```

```

        "correct_answer": "...",
        "topic_id": "topic-2",
        "q_vector": [0, 0, 1, 1, 0, 0, 0, 0],
        "attribute_count": 2
    }
],
"pagination": {
    "total": 25,
    "page": 1,
    "page_size": 10,
    "total_pages": 3,
    "has_more": true
}
}

```

Use Cases

1. Machine Learning / CDM Models

The `q_vector` format is perfect for Cognitive Diagnostic Models:

```

import numpy as np

# Extract Q-matrix for all questions
response =
requests.get('/api/hierarchy/chapter/{id}/questions/enhanced').json()
attributes = response['attributes']
questions = response['questions']

# Build Q-matrix as numpy array
Q = np.array([q['q_vector'] for q in questions])

# Q is now a (n_questions x n_attributes) binary matrix
print(f"Q-matrix shape: {Q.shape}")
# Output: Q-matrix shape: (25, 8)

# Use in DINA, DINO, or other CDM models
from pyirt import irt
model = irt.IRT(Q, response_matrix)
model.fit()

```

2. Attribute Coverage Analysis

```

# Check which attributes are well-covered
attribute_coverage = Q.sum(axis=0)

for idx, attr in enumerate(attributes):

```

```
count = attribute_coverage[idx]
print(f"{attr['name']}: {count} questions")
```

3. Question Similarity

```
# Find similar questions based on attribute overlap
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(Q)

# Find questions similar to question 0
similar_indices = np.argsort(similarity_matrix[0])[:, -1][1:6]
print(f"Questions similar to Q0: {similar_indices}")
```

4. Frontend Display

```
// Display questions with their attributes
const response = await
fetch(`/api/hierarchy/chapter/${chapterId}/questions/enhanced`);
const data = await response.json();

data.questions.forEach(question => {
  console.log(`Question: ${question.content}`);

  // Show which attributes this question tests
  const testedAttributes = data.attributes.filter((attr, idx) =>
    question.q_vector[idx] === 1
  );

  console.log('Tests attributes:', testedAttributes.map(a => a.name));
});
```

Comparison with Standard Endpoint

Standard Endpoint

```
GET /api/hierarchy/{level}/{item_id}/questions
```

Returns: Questions with attributes embedded in each question

Use Case: General question browsing, simple UI display

Enhanced Endpoint

```
GET /api/hierarchy/{level}/{item_id}/questions/enhanced
```

Returns: Questions with consolidated attribute list + binary vectors

Use Case: Analytics, machine learning, attribute analysis

Supported Hierarchical Levels

Competitive Exam Path

```
exam → subject → chapter → topic
```

Examples:

- `/api/hierarchy/exam/{exam_id}/questions/enhanced` - All questions in exam
- `/api/hierarchy/subject/{subject_id}/questions/enhanced` - All questions in subject
- `/api/hierarchy/chapter/{chapter_id}/questions/enhanced` - All questions in chapter
- `/api/hierarchy/topic/{topic_id}/questions/enhanced` - All questions in topic

School Path

```
exam → class → subject → chapter → topic
```

Examples:

- `/api/hierarchy/class/{class_id}/questions/enhanced` - All questions for a class
 - `/api/hierarchy/subject/{subject_id}/questions/enhanced` - All questions in subject
 - `/api/hierarchy/chapter/{chapter_id}/questions/enhanced` - All questions in chapter
-

Performance Considerations

Optimization Tips

1. **Use pagination** for large result sets
2. **Cache attribute lists** at each level (they don't change frequently)
3. **Fetch only needed pages** rather than all questions at once

Response Size

For a chapter with:

- 100 questions
- 15 attributes
- Each `q_vector` has 15 elements

Approximate size: ~50-100 KB per page (depending on question content)

Error Responses

Invalid Level

```
{
  "error": "Invalid level. Must be one of: ['exam', 'subject', 'chapter', 'topic', 'class']"
}
```

Status: 400

Item Not Found

Questions array will be empty if no questions exist at that level.

```
{
  "level": "chapter",
  "level_id": "...",
  "total_questions": 0,
  "attributes": [],
  "questions": [],
  "pagination": {...}
}
```

Status: 200

Advanced Examples

Get Q-Matrix for Entire Exam

```
# Fetch all questions with their attribute vectors
curl
"http://localhost:5200/api/hierarchy/exam/{exam_id}/questions/enhanced?
page_size=1000"
```

Filter by Topic and Get Vectors

```
# Get questions from a specific topic
curl
"http://localhost:5200/api/hierarchy/topic/{topic_id}/questions/enhanced"
```

Batch Processing

```
import requests

def get_all_questions_with_vectors(level, level_id):
    """Fetch all questions with pagination."""
    all_questions = []
    page = 1

    while True:
        response = requests.get(
            f'/api/hierarchy/{level}/{level_id}/questions/enhanced',
            params={'page': page, 'page_size': 100}
        ).json()

        all_questions.extend(response['questions'])

        if not response['pagination']['has_more']:
            break

        page += 1

    return {
        'attributes': response['attributes'],
        'questions': all_questions
    }

# Usage
data = get_all_questions_with_vectors('chapter', 'chapter-uuid')
print(f"Total questions: {len(data['questions'])}")
print(f"Total attributes: {len(data['attributes'])}")
```

Integration with CDM Libraries

pyirt (Python)

```
import requests
import numpy as np
from pyirt import irt

# Fetch data
response = requests.get('/api/hierarchy/chapter/{id}/questions/enhanced').json()

# Build Q-matrix
Q = np.array([q['q_vector'] for q in response['questions']])

# Build response matrix (student responses)
# responses[i][j] = 1 if student i answered question j correctly
```



```
responses = get_student_responses() # Your function

# Fit IRT model
model = irt.IRT(Q, responses)
model.fit()

# Get student knowledge states
knowledge_states = model.get_alpha()
```

R (CDM package)

```
library(httr)
library(jsonlite)
library(CDM)







# Fetch data
response <-
GET("http://localhost:5200/api/hierarchy/chapter/{id}/questions/enhanced")
data <- fromJSON(content(response, "text"))

# Build Q-matrix
Q <- do.call(rbind, lapply(data$questions, function(q) q$q_vector))

# Fit DINA model
responses <- get_student_responses() # Your function
model <- din(responses, Q)
summary(model)
```

Summary

The enhanced questions endpoint provides:

-  Consolidated attribute list for the hierarchical level
-  Binary Q-vectors for each question
-  Efficient format for ML/CDM models
-  Pagination support
-  Works with both competitive and school paths
-  Compatible with standard CDM libraries

Key Benefit: No need to manually build Q-matrices - the API provides them ready for analysis!