

버전: 3.0 (모듈화 + AI Judge 재시도 통합) 업데이트: 2025-10-02 기술 스택: Gemini 2.5 Pro (생성) + GPT-4o-mini (검증)

국어 문제 생성 시스템은 **4가지 유형**(시, 소설, 수필/비문학, 문법)에 대해 각각 최적화된 파이프라인을 제공합니다.

- 영어 프롬프트 기반 - LLM 성능 최적화
- 2단계 검증 시스템 - Gemini (구조) + GPT-4o-mini (내용)
- 병렬 처리 - 작품별 동시 생성 (시/소설/수필)
- 유형별 맞춤 전처리 - AI 핵심 발체, 영역 분할 등
- AI Judge 재시도 메커니즘 - invalid 문제 자동 재생성
- 모듈화 아키텍처 - validators/, utils/ 분리

```

[사용자 요청]
↓
tasks.py (Celery 비동기 작업)
↓
generate_problems_parallel() [병렬 처리 엔트리]
↓

```

시/소설/수필	문법
↓	↓
[병렬 생성]	[순차 생성]
ThreadPoolExecutor	영역별 생성
max_workers=5	I~V 순서대로
↓	↓

```

↓
_generate_problems_for_work_parallel()
↓
_generate_multiple_problems_from_single_text()
[AI Judge 재시도 로직 포함, max_retries=3]
↓

```

Loop: attempt 1 ~ max_retries
-------------------------------

```

↓
[1] Gemini 2.5 Pro로 문제 생성 (배치)
↓
[2] JSON 파싱 및 구조 변환
↓

```

```
[3] AI Judge 검증 (각 문제별)
    validators/ai_judge_validator.py
    GPT-4o-mini로 4가지 기준 평가
    ↓
    └─ VALID → valid_problems 리스트 누적
    └─ INVALID → invalid_problems 리스트에 피드백 저장
    ↓
[4] 목표 개수 달성 체크
    len(valid_problems) >= count?
    └─ YES → 반환 (성공)
    └─ NO → 계속
    ↓
[5] 피드백으로 프롬프트 재구성
    _rebuild_korean_prompt_with_feedback()
    invalid 문제들의 점수와 피드백 추가
    ↓
[6] 다음 시도 (attempt + 1)
    ───────────
    ↓
    최종: valid_problems 반환
    ↓
    [DB 저장] - tasks.py에서 처리
```

## 난이도 체계

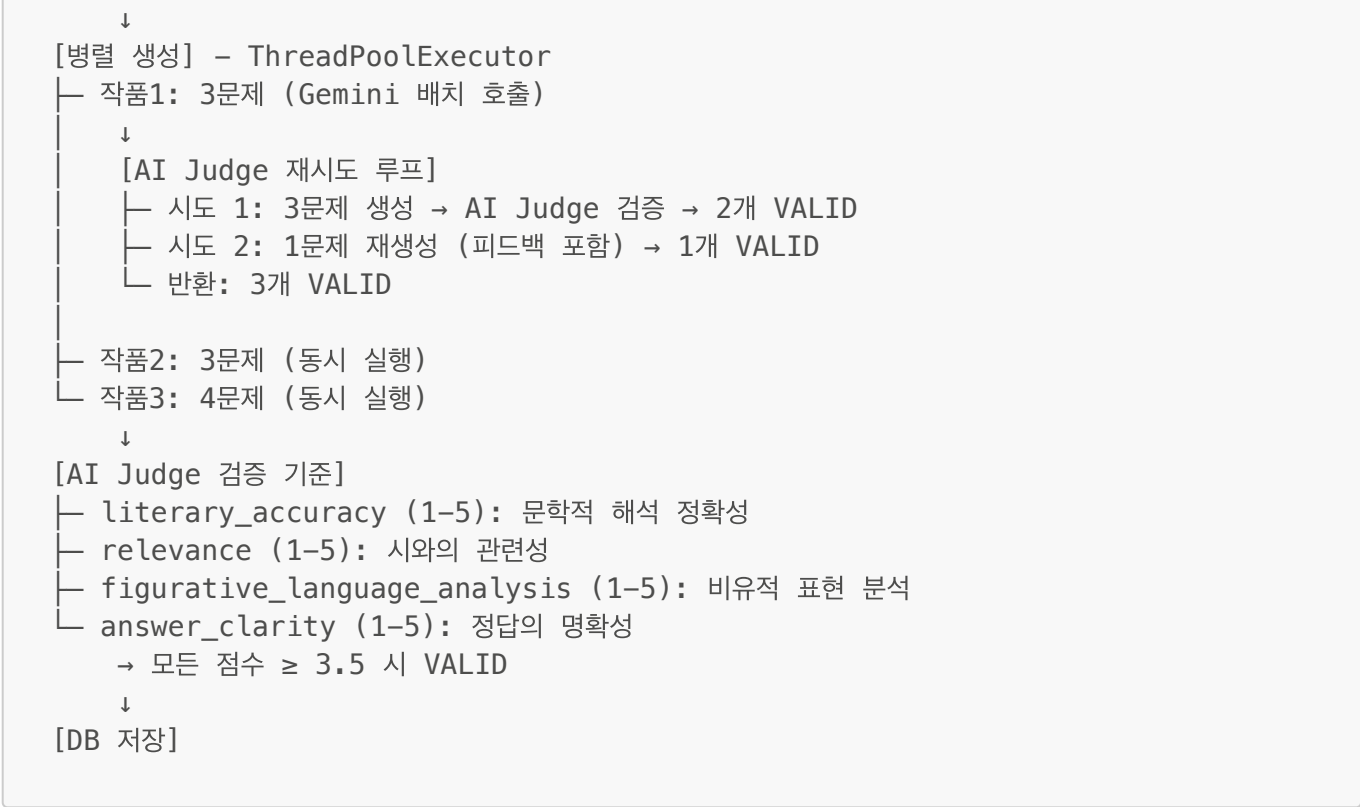
모든 유형에 공통 적용되는 난이도 기준:

난이도	영문	설명	특징
상	Advanced	깊은 분석, 추론, 복잡한 사고 필요	고급 문학 기법, 비판적 사고, 통합 분석
중	Intermediate	중간 수준의 이해와 해석 필요	인과관계, 주제 파악, 기법 분석
하	Basic	표면적 이해와 회상 필요	직접 정보, 기본 이해, 명시적 내용

## 1. 시 (Poetry)

플로우

```
요청: 시 10문제 생성
↓
[작품 선택 및 병렬화]
└─ 작품 수 결정: 3개 (10문제 ≤ 10)
└─ data/poem/*.txt 에서 랜덤 선택
└─ 파일명 파싱: "제목-작가.txt"
└─ 문제 수 분배: [3, 3, 4]
↓
[전처리]
└─ 시는 전체 텍스트 사용 (일반적으로 짧음)
└─ 2000자 초과 시 앞부분만 사용 (연작시 대응)
```



난이도별 출제 기준

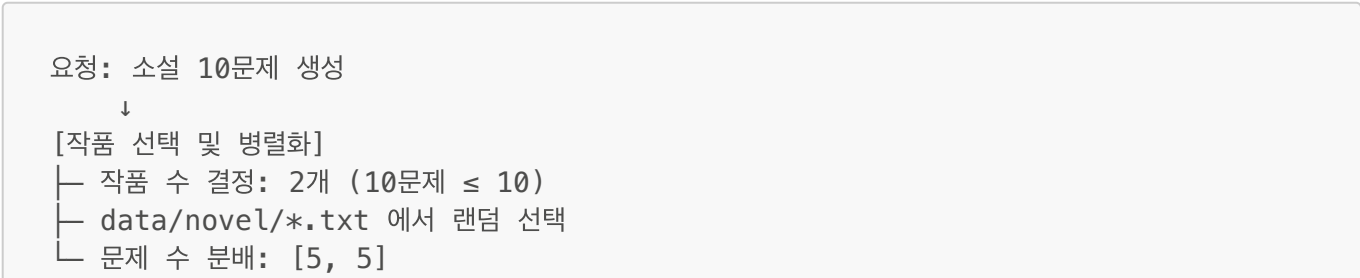
난이도	출제 초점	예시
하	표면적 이해, 화자 파악, 기본 정서, 직접적 이미지	"시의 화자는 누구인가?", "시에서 반복되는 단어는?"
중	비유적 표현, 어조, 분위기, 주제 요소, 시적 장치	"이 시의 주된 시적 기법은?", "3연의 분위기는?"
상	깊은 상징적 의미, 복잡한 문학 기법, 작가 의도, 비교 분석	"상징적 의미는?", "시적 화자의 내면 변화는?"

주요 출제 영역

- **문학 장치:** 은유(Metaphor), 의인법(Personification), 직유(Simile)
- **표현 기법:** 이미지(Imagery), 상징(Symbolism), 어조(Tone)
- **구조 요소:** 운율(Rhythm), 반복(Repetition), 대조(Contrast)

2. 소설 (Novel/Fiction)

플로우





난이도별 출제 기준

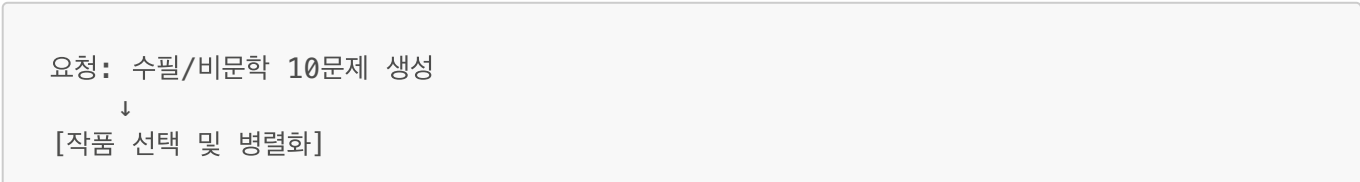
난이도	출제 초점	예시
하	플롯 순서, 인물 파악, 배경, 직접적 대화 해석	"사건의 순서는?", "주인공은 누구인가?"
중	인물 동기, 갈등 분석, 서술 관점, 인과관계	"인물의 행동 동기는?", "갈등의 원인은?"
상	주제의 깊이, 서사 기법, 심리적 복잡성, 상징적 해석	"작품의 주제 의식은?", "서술자의 시점 변화 효과는?"

주요 출제 영역

- **서사 요소**: 플롯, 인물, 배경, 갈등
- **서술 기법**: 시점, 서술자, 플래시백, 복선
- **인물 분석**: 심리, 동기, 관계, 성격 변화

3. 수필/비문학 (Essay/Non-fiction)

플로우



```
└─ 작품 수 결정: 2개 (10문제 ≤ 10)
└─ data/non-fiction/*.txt 에서 랜덤 선택
└─ 문제 수 분배: [5, 5]
    ↓
[전처리 - AI 핵심 발췌]
└─ 1500자 초과 시 핵심 부분 추출
└─ 발췌 기준 (영어 프롬프트):
    └─ Main argument (핵심 주장)
    └─ Key evidence (주요 증거)
    └─ Central thesis (중심 논지)
    └─ Author's main point (저자의 주요 논점)
└─ 목표 길이: 800-1200자
└─ 실패 시 폴백: 원본 앞 1200자
    ↓
[병렬 생성] - ThreadPoolExecutor
└─ 작품1: 5문제 (발췌된 핵심 논지로)
    ↓
    [AI Judge 재시도 루프, max_retries=3]
└─ 작품2: 5문제 (동시 실행)
    ↓
[AI Judge 검증 기준]
└─ argument_comprehension (1-5): 논증 이해도
└─ relevance (1-5): 지문과의 관련성
└─ critical_thinking (1-5): 비판적 사고
└─ answer_clarity (1-5): 정답의 명확성
    ↓
[DB 저장]
```

난이도별 출제 기준

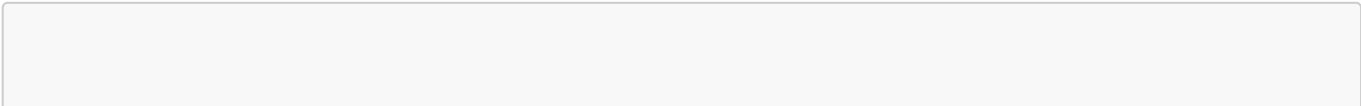
난이도	출제 초점	예시
하	주제 파악, 명시적 정보 회상, 기본 구조	"글의 중심 내용은?", "글쓴이의 직업은?"
중	논증 분석, 증거 평가, 논리적 관계, 글쓴이의 목적	"주장을 뒷받침하는 근거는?", "논리 전개 방식은?"
상	비판적 평가, 추론, 새로운 맥락 적용, 수사적 분석	"글쓴이의 관점에 대한 비판은?", "다른 상황 적용은?"

주요 출제 영역

- 논지 파악: 중심 주장, 근거/논거, 논리 전개
- 글의 구조: 서론-본론-결론, 글의 조직
- 글쓴이 의도: 관점, 목적, 태도

4. 문법 (Grammar)

플로우



```

요청: 문법 10문제 생성
↓
[문법 영역 분할] - 병렬 처리 없음
├ data/grammar.txt 단일 파일 로드
└ I~V 영역으로 분할:
  ├── I. 음운 (Phonology)
  ├── II. 품사와 어휘 (Morphology/Lexicon)
  ├── III. 문장 (Syntax)
  ├── IV. 기타 (Miscellaneous)
  └ V. 부록 (Appendix)
↓
[문제 수 균등 분배]
└ 10문제 ÷ 5영역 = [2, 2, 2, 2, 2]
↓
[순차 생성] - 영역별 (병렬 X)
├ I. 음운: 2문제 생성
│   └ [AI Judge 재시도 루프, max_retries=3]
│       ├── 새로운 예문 생성 가능
│       ├── grammar.txt 원본 참조 금지
│       └ LLM이 문법 개념에 맞는 예문 작성
├ II. 품사와 어휘: 2문제
├ III. 문장: 2문제
├ IV. 기타: 2문제
└ V. 부록: 2문제
↓
폴백 메커니즘
└ 영역별 생성 실패 시 → 개별 생성
↓
[AI Judge 검증 기준]
├ grammar_accuracy (1-5): 문법 개념의 정확성
├ example_quality (1-5): 예문의 품질
├ explanation_clarity (1-5): 설명의 명확성
└ answer_clarity (1-5): 정답의 명확성
↓
[source_text 처리]
├ LLM 생성 예문 → 문제에 표시
└ grammar.txt 원본 → 숨김 (표시 안 함)
↓
[DB 저장]

```

## 난이도별 출제 기준

난이도	출제 초점	예시
하	기본 문법 용어, 단순 규칙, 명확한 예시	"주어의 역할은?", "명사의 정의는?"
중	문법 규칙 적용, 문장 구조 분석, 품사 식별	"문장 성분 분석", "용언의 활용형 구별"
상	복잡한 문법 개념, 예외 규칙, 비교 분석, 언어학적 추론	"음운 변동의 조건", "문법 범주의 상관관계"

## 주요 출제 영역

- 음운론 (Phonology): 음운 변동, 발음 규칙
- 형태론 (Morphology): 품사, 단어 형성
- 통사론 (Syntax): 문장 구조, 문장 성분
- 의미론 (Semantics): 의미 관계, 어휘 관계

## 문법 특이사항

1. 병렬 처리 없음 - 영역별 순차 생성 (I → II → III → IV → V)
2. 새 예문 생성 - LLM이 문법 개념에 맞는 새로운 예문 작성 가능
3. 원본 숨김 - grammar.txt 원본 내용은 문제에 표시하지 않음
4. 영역별 폴백 - 각 영역 생성 실패 시 개별 생성 모드로 전환

---

## AI Judge 재시도 메커니즘 (핵심)

### 개요

모든 문제 생성 시 AI Judge 검증을 통과한 문제만 반환하도록 자동 재시도합니다.

### 로직 상세

```
def _generate_multiple_problems_from_single_text(..., max_retries=3):
    """하나의 지문으로 여러 문제 생성 (AI Judge 재시도 포함)"""

    valid_problems = [] # 합격한 문제 누적
    original_prompt = ... # 초기 프롬프트

    for attempt in range(max_retries): # 최대 3회 시도
        needed_count = count - len(valid_problems)
        if needed_count <= 0:
            return valid_problems[:count] # 목표 달성

        # [1] Gemini로 문제 생성
        response = self.model.generate_content(prompt)
        problems = self._parse_and_validate_problems(...)

        # [2] AI Judge 검증
        invalid_problems = []
        for problem in problems:
            is_valid, scores, feedback =
self.ai_judge_validator.validate_problem(
    problem, korean_type
)

            if is_valid:
                valid_problems.append(problem) # 합격 누적
            else:
                invalid_problems.append({
                    "problem": problem,
```

```
        "feedback": feedback,
        "scores": scores
    })

    # [3] 목표 달성 체크
    if len(valid_problems) >= count:
        return valid_problems[:count]

    # [4] 재시도용 프롬프트 재구성
    if attempt < max_retries - 1 and invalid_problems:
        prompt = self._rebuild_korean_prompt_with_feedback(
            original_prompt, invalid_problems, korean_type
        )

    return valid_problems # 부족하더라도 생성된 문제 반환
```

## 피드백 프롬프트 예시

```
**IMPORTANT: Previous attempt had validation failures. Fix these issues:**

Problem 1 feedback:
- Scores: literary_accuracy=3.0, relevance=4.0,
  figurative_language_analysis=3.2, answer_clarity=4.5
- Issue: 비유적 표현 분석이 부족합니다. 시의 은유를 더 깊이 있게 다뤄야 합니다.

Problem 2 feedback:
- Scores: narrative_comprehension=3.5, relevance=3.2,
  textual_analysis=3.0, answer_clarity=4.0
- Issue: 지문과의 관련성이 약합니다. 본문 내용을 직접 활용한 문제를 출제하세요.

**MUST ensure**: All scores >= 3.5, answer_clarity >= 4.0, relevance to
source text
```

## 재시도 통계

시도	목표	생성	VALID	INVALID	누적 VALID
1차	5개	5개	3개	2개	3개
2차	2개	2개	1개	1개	4개
3차	1개	1개	1개	0개	5개 (완료)

## 단일 문제 재생성

### 엔드포인트

```
POST /api/korean-generation/regenerate-problem/{problem_id}
```



## 플로우

요청: 문제 ID + 개선 요구사항

↓

[1] 기존 문제 데이터 로드

```
├ korean_type
├ source_text, source_title, source_author
├ question, choices, correct_answer, explanation
└ difficulty
```

↓

[2] 영어 재생성 프롬프트 구성

SingleProblemEnglishTemplate 사용

```
├ System: "You are an expert Korean language teacher"
├ Current Problem: 기존 문제 모든 정보 전달
├ Requirements: 사용자 요구사항 (한글 또는 영어)
└ Instruction: "Regenerate improved problem based on requirements"
```

↓

[3] Gemini 2.5 Pro 호출

└ 한 번의 호출로 개선된 문제 반환

↓

[4] JSON 파싱

```
{
  "question": "...",
  "choices": ["A", "B", "C", "D"],
  "correct_answer": "A",
  "explanation": "..."
}
```

↓

[5] 기존 문제 업데이트

```
├ question → 새 질문
├ choices → 새 선택지
├ correct_answer → 새 정답
└ explanation → 새 해설
```

↓

[6] DB 저장 및 반환

## 재생성 프롬프트 예시

""""

You are an expert Korean language teacher specializing in poem analysis.

Current Problem:

- Type: 시 (Poem)
- Source: "진달래꽃" by 김소월
- Question: 이 시의 주된 정서는?
- Choices:
  - A. 기쁨
  - B. 슬픔
  - C. 분노

```
D. 희망
- Correct Answer: B
- Explanation: 시적 화자는...

User Requirements:
"선택지를 더 구체적으로 만들어주세요. 감정보다는 시적 기법에 초점을 맞춰주세요."

Task:
Regenerate an improved problem addressing the user's requirements.
Keep the same source text and general theme.
ALL output content MUST be in KOREAN.

Return ONLY valid JSON:
{
  "question": "...",
  "choices": ["...", "...", "...", "..."],
  "correct_answer": "A/B/C/D",
  "explanation": "..."
}
""""
```

특징

- **AI Judge 미적용:** 재생성은 사용자 직접 요청이므로 바로 저장
- **1회 호출:** 재시도 없이 단일 Gemini 호출
- **컨텍스트 유지:** source\_text, 난이도, 유형 모두 유지
- **유연한 요구사항:** 한글/영어 모두 지원

검증 기준 요약

AI Judge 검증 기준 (GPT-4o-mini)

유형	기준 1	기준 2	기준 3	기준 4
시	literary_accuracy	relevance	figurative_language_analysis	answer_clarity
소설	narrative_comprehension	relevance	textual_analysis	answer_clarity
수필/ 비문학	argument_comprehension	relevance	critical_thinking	answer_clarity
문법	grammar_accuracy	example_quality	explanation_clarity	answer_clarity

합격 기준: 모든 항목 ≥ 3.5/5.0

공통 구조 검증

- 필수 필드: question, choices(4개), correct\_answer, explanation, difficulty
- 정답 형식: A/B/C/D
- 선택지: 4개, 중복 없음
- 난이도: 상/중/하

## 기술 스펙

### LLM 모델

- 생성: Gemini 2.5 Pro (Google)
- 검증: GPT-4o-mini (OpenAI)

### 프롬프트 언어

- 입력: 영어 (English-based prompts)
- 출력: 한국어 (Korean language content)

### 병렬 처리

- 엔진: ThreadPoolExecutor
- 최대 워커: min(작품 수, 5)
- 적용 유형: 시, 소설, 수필/비문학
- 미적용 유형: 문법 (순차 처리)

### 재시도 메커니즘

- 최대 시도: 3회 (max\_retries=3)
- 대기 시간: 1초 (time.sleep(1))
- 적용 범위: 모든 생성 메서드 (AI Judge 재시도)
- 재생성 제외: 단일 문제 재생성은 재시도 없음

## 성능 최적화

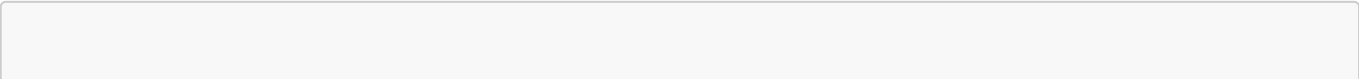
### 작품 수 자동 결정

문제 수	시	소설	수필/비문학
≤ 10	3개	2개	2개
≤ 20	6개	4개	4개
> 20	min(문제수 ÷ 3, 10)개	min(문제수 ÷ 3, 10)개	min(문제수 ÷ 3, 10)개

### 발체 전략

- 시: 2000자 이하 → 전체 사용
- 소설: 1500자 초과 → AI 핵심 발체 (갈등/대화 중심)
- 수필/비문학: 1500자 초과 → AI 핵심 발체 (논지/증거 중심)
- 문법: 영역별 분할 (I~V)

## 모듈 구조



```
backend/services/korean-service/app/
├── services/
│   ├── korean_problem_generator.py    # 메인 생성 로직 (711줄)
│   ├── validators/
│   │   ├── __init__.py
│   │   └── ai_judge_validator.py      # AI Judge 검증 (130줄)
│   └── utils/
│       ├── __init__.py
│       └── retry_handler.py          # 재시도 유틸리티
├── prompt_templates/
│   ├── base_template_en.py           # 영어 기반 템플릿
│   ├── single_problem_en.py          # 단일 문제 생성
│   └── multiple_problems_en.py        # 다중 문제 생성
├── routers/
│   └── korean_generation.py           # API 엔드포인트
└── tasks.py                           # Celery 비동기 작업
```

---