Full Technical Appendix

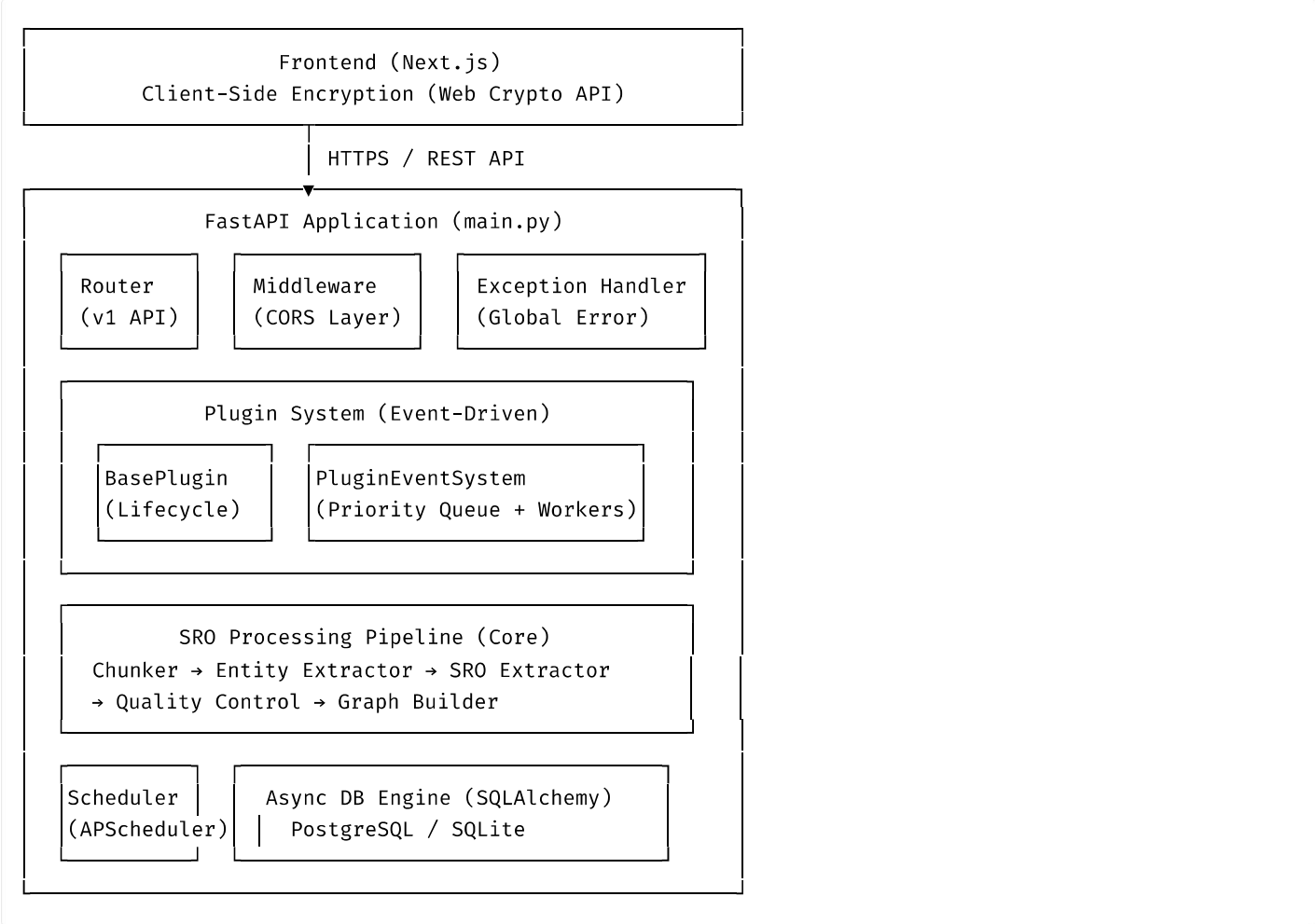
KRU Platform / SRO App V2.8

> เอกสารทางเทคนิคฉบับสมบูรณ์สำหรับคณะกรรมการเทคนิคและนักลงทุน (Venture Capital) จัดทำจากการวิเคราะห์ Source Code และ Database Schema โดยตรง

PAGE 1: SYSTEM ARCHITECTURE & ENGINE TOPOLOGY

1.1 สถาปัยกรรมระบบภาพรวม

ระบบ SRO App V2 ออกแบบบนสถาปัยกรรม **Async-First Backend** โดยใช้ **FastAPI** เป็นเฟรมเวิร์กหลัก รองรับการทำงานแบบ Asynchronous I/O ทุกชั้น ตั้งแต่ Database Session ไปจนถึง Plugin Message Processing



1.2 Communication Patterns

1.2.1 Asynchronous Request Handling

ระบบใช้ `async/await` ทุก Endpoint โดย FastAPI ทำงานบน `uvicorn` (ASGI Server) ซึ่งรองรับ Concurrent Request โดยไม่ Block I/O

- **Database Session**: ใช้ `AsyncSession` จาก `sqlalchemy.ext.asyncio` พร้อม `async_sessionmaker` ที่ตั้งค่า `expire_on_commit=False` และ `autoflush=False` เพื่อลด Overhead
- **Connection Pool**: กำหนด `pool_size` , `max_overflow` , `pool_timeout` , และ `pool_recycle` แบบไดนามิกตาม Environment

1.2.2 Event-Driven Plugin Hooks

ระบบ Plugin ออกแบบเป็น **Event-Driven Architecture** โดยใช้ `PluginEventSystem` ซึ่งประกอบด้วย:

ส่วนประกอบ	หน้าที่	Implementation
Event Queue	Priority-based Queue สำหรับ Event ที่รอประมวลผล	`asyncio.PriorityQueue` พร้อม Event Priority 5 ระดับ
Subscriptions	การสมัครรับ Event ตามประเภท	`EventSubscription` รองรับ Event Type Filter และ Handler Callback
Event Workers	Worker Tasks ประมวลผล Event แบบ Async	ทำงานเป็น Background Task พร้อม Retry Mechanism
Analytics	การติดตามและวัดผล Event Processing	บันทึก Emitted/Processed/Failed Counts และ Event History

Plugin Lifecycle States (10 สถานะ):

Event Types ที่รองรับ: Document Processing, SRO Extraction, Knowledge Graph Updates, Plugin Lifecycle, System Events, Configuration Changes, Performance Metrics

1.2.3 Plugin Message Passing

`BasePlugin` รองรับการสื่อสารระหว่าง Plugin ผ่าน **Internal Async Message Queue**:

- **PluginMessage**: Dataclass สำหรับส่งข้อความระหว่าง Plugin พร้อม Message Type, Priority, และ Payload
- **Message Handlers**: ลงทะเบียน Handler ตาม Message Type โดยใช้ `_register_message_handlers()`
- **Event Listeners**: ลงทะเบียน Listener สำหรับ Event เฉพาะโดยใช้ `_register_event_listeners()`
- **Message Processing Loop**: `asyncio.create_task(self._process_messages())` ทำงานเป็น Background Coroutine

1.2.4 Scheduled Tasks (Cron-Based)

ระบบใช้ `APScheduler` สำหรับ Cron Jobs:

- **Curriculum Sync**: ทำงานทุกวันอาทิตย์ เวลา 02:00 น. (CronTrigger: `day_of_week='sun', hour=2, minute=0`) สำหรับ Sync มาตรฐานหลักสูตร OBEC

1.3 Intelligent Request Queue System

ระบบจัดการ Request Queue อัจฉริยะตาม Configuration ใน `config/system.yaml`:

Parameter	ค่า (Production)	รายละเอียด
Max TPM	4,000,000 tokens/min	Token Per Minute Limit
Target TPM	3,200,000 (80%)	Conservative Target
Circuit Breaker Failure Threshold	5	จำนวนความล้มเหลวก่อนเปิด Circuit
Recovery Timeout	60 วินาที	เวลารอก่อนลอง Recovery
Max Concurrent Tasks	10	จำนวน Task ที่ทำงานพร้อมกันสูงสุด
Max Queue Size	100	ความจุ Queue สูงสุด
Max Retries	3	จำนวนครั้ง Retry สูงสุด
Exponential Backoff Base	2 วินาที	Base สำหรับ Backoff Calculation

Priority Levels: CRITICAL (0) > HIGH (1) > MEDIUM (2) > LOW (3) > BACKGROUND (4)

Circuit Breaker States: Closed → Open (failure threshold exceeded) → Half-Open (recovery attempt) → Closed

Fallback Strategy: QUEUE (รอใน Queue เมื่อ Circuit เปิด)

1.4 Pipeline Execution Topology

`SROProcessingPipeline` ทำงานเป็น **6-Stage Sequential Pipeline** โดยแต่ละ Stage รองรับ Parallel Processing ภายใน:

```
Stage 1: CHUNKING (Adaptive Chunker)
    ↓
Stage 2: ENTITY_EXTRACTION (Parallel Batch Processing)
    ↓
Stage 3: SRO_EXTRACTION (Parallel Batch Processing)
    ↓
Stage 4: QUALITY_CONTROL (5-Metric Evaluation)
    ↓
Stage 5: GRAPH_BUILDING (NetworkX DiGraph Construction)
    ↓
Stage 6: FINALIZATION (Result Aggregation)
```

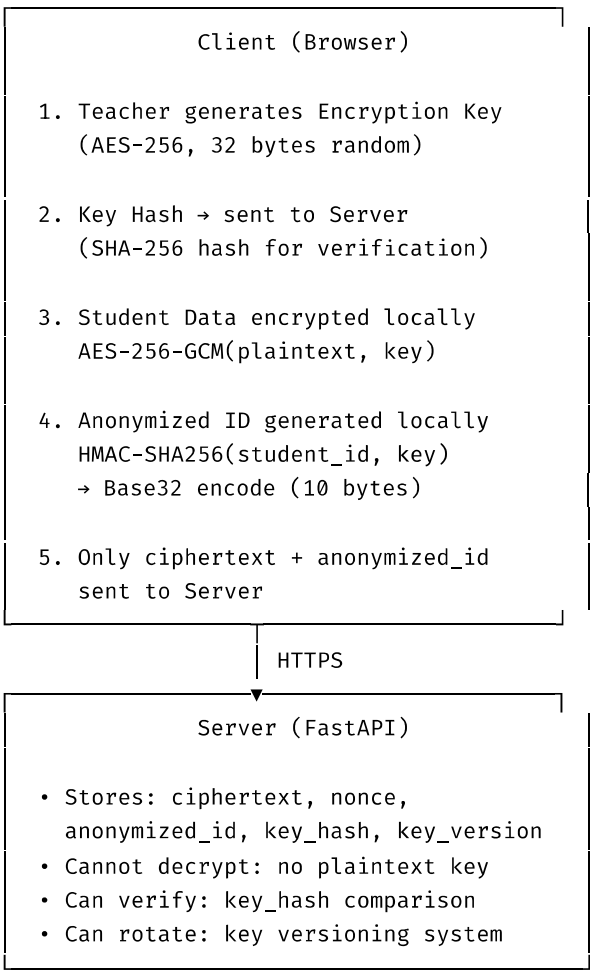
PipelineManager จัดการ Concurrent Pipelines สูงสุด 5 อินสแตนซ์พร้อมกันโดยใช้ `asyncio.Semaphore`

ระบบรองรับ **Pause/Resume/Cancel** ผ่าน `asyncio.Event` สำหรับ Pipeline Control แบบ Real-time

PAGE 2: ZERO-KNOWLEDGE SECURITY & PDPA SPECIFICATIONS

2.1 สถาปัยกรรม Zero-Knowledge

ระบบออกแบบตามหลัก **Zero-Knowledge Architecture** โดยข้อมูล PII (Personally Identifiable Information) ของนักเรียนถูกเข้ารหัสที่ **Client-Side** ก่อนส่งไปยัง Server เซิร์ฟเวอร์ไม่สามารถถอดรหัสชื่อจริงของนักเรียนได้ เนื่องจาก Key ถูกเก็บไว้ที่ฝั่ง Client เท่านั้น



2.2 การเข้ารหัส AES-256-GCM

2.2.1 Backend EncryptionService

Parameter	ค่า	รายละเอียด
Algorithm	AES-256-GCM	Authenticated Encryption with Associated Data
Key Length	32 bytes (256 bits)	`os.urandom(32)`
Nonce Length	12 bytes (96 bits)	NIST Recommended for GCM
Key Derivation	HKDF-SHA256	HMAC-based Key Derivation Function

Parameter	ค่า	รายละเอียด
Salt Length	16 bytes	`os.urandom(16)`
Info Tag	`b'sro-student-encryption'`	Context-specific Info for HKDF

การทำงานของ Encryption Flow:

1. **Key Generation**: `os.urandom(32)` สร้าง Random Key 256-bit
2. **Key Derivation**: ใช้ HKDF กับ Master Key + Salt + Info Tag เพื่อ derive Child Key
3. **Encryption**: `AESGCM(key).encrypt(nonce, plaintext, None)` ส่งผลลัพธ์เป็น `{ciphertext, nonce}` ในรูป Base64
4. **Decryption**: `AESGCM(key).decrypt(nonce, ciphertext, None)` ถอดรหัสกลับเป็น Plaintext
5. **Key Hashing**: `hashlib.sha256(key).hexdigest()` สำหรับเก็บใน DB เพื่อ Verification โดยไม่เก็บ Key จริง

2.2.2 Frontend ClientEncryption (Web Crypto API)

ฝั่ง Frontend ใช้ **Web Crypto API** ของ Browser สำหรับการเข้ารหัส โดยไม่ต้องพึ่งพา Library ภายนอก:

- **Key Import**: `crypto.subtle.importKey('raw', keyBuffer, {name: 'AES-GCM'}, false, ['encrypt'])`
- **Encryption**: `crypto.subtle.encrypt({name: 'AES-GCM', iv: nonce}, cryptoKey, data)` โดยใช้ Nonce 12 bytes จาก `crypto.getRandomValues()`
- **Output**: `{ciphertext, nonce}` ในรูป Base64 (เหมือนกับ Backend)

2.3 การสร้าง Anonymized ID ด้วย HMAC-SHA256

ระบบสร้าง **Deterministic Anonymized ID** จากรหัสนักเรียนเพื่อให้สามารถเชื่อมโยงข้อมูลได้โดยไม่เปิดเผยตัวตน:

Backend Implementation:

```
hmac_obj = hmac.new(key, student_id.encode('utf-8'), hashlib.sha256)
hash_digest = hmac_obj.digest()
anonymized_id = base64.b32encode(hash_digest[:10]).decode('utf-8').upper()
```

Frontend Implementation:

```
const cryptoKey = await crypto.subtle.importKey('raw', keyBuffer,
    {name: 'HMAC', hash: 'SHA-256'}, false, ['sign']);
const signature = await crypto.subtle.sign('HMAC', cryptoKey, dataBuffer);
const bytes = new Uint8Array(signature).slice(0, 10);
return base32Encode(bytes);
```

- ใช้ Key เดียวกัน → ได้ Anonymized ID เดียวกันเสมอ (Deterministic)
- ใช้ 10 bytes แรกของ HMAC-SHA256 Digest → 80 bits of entropy
- เข้ารหัสด้วย Base32 → 16 ตัวอักษร ที่อ่านง่ายและ Case-insensitive

2.4 Key Management & Rotation

2.4.1 Teacher Encryption Keys

Field	Type	รายละเอียด
`teacher_id`	UUID (FK → teachers.id)	เชื่อมกับครูผู้สร้าง Key
`key_hash`	String(64)	SHA-256 Hash ของ Key จริง
`key_version`	Integer	Version ของ Key (เริ่มที่ 1)
`is_active`	Boolean	สถานะ Key ปัจจุบัน
`expires_at`	DateTime	วันหมดอายุของ Key
`created_by`	UUID (FK → users.id)	ผู้สร้าง Key

2.4.2 School Encryption Keys

โครงสร้างเดียวกับ Teacher Encryption Keys แต่เชื่อมกับ `schools.id` สำหรับระดับโรงเรียน

2.4.3 Key Migration

`KeyMigrationLog` บันทึกการเปลี่ยน Key Type พร้อม Audit Trail:

- `migration_type` : ประเภทการ Migration
- `from_key_type / to_key_type` : Key Type เดิมและใหม่
- `students_migrated` : จำนวนนักเรียนที่ถูก Migrate
- `migrated_by` : ผู้ทำการ Migrate
- `reason` : เหตุผลในการ Migrate

2.5 PDPA Consent Management

`TeacherConsent` บันทึกการให้ความยินยอมตามพระราชบัญญัติคุ้มครองข้อมูลส่วนบุคคล (PDPA):

Field	Type	รายละเอียด
`teacher_id`	UUID (FK → users.id)	ครูผู้ให้ความยินยอม
`action_type`	String(50)	ประเภทการกระทำ (เช่น upload, data_processing)
`classroom_token`	String(20)	ตัวระบุห้องเรียน
`academic_year`	String(10)	ปีการศึกษา
`ip_address`	String(45)	IP Address ของผู้ให้ความยินยอม
`user_agent`	Text	Browser User Agent
`timestamp`	DateTime (UTC)	เวลาที่ให้ความยินยอม

2.6 Client-Side Upload Flow (Zero-Knowledge Verification)

จากการวิเคราะห์ `student-upload.tsx` กระบวนการอัปโหลดมีขั้นตอนดังนี้:

1. **Key Check**: ตรวจสอบว่าครูมี Encryption Key หรือไม่ (`hasTeacherEncryptionKey()`)
2. **Consent Modal**: แสดง `ZeroKnowledgeModal` ก่อนอัปโหลดครั้งแรก (PDPA Compliance)
3. **CSV Parsing**: อ่านไฟล์ CSV รายชื่อนักเรียน (student_id, national_id, real_name)
4. **Client-Side Encryption**: สำหรับนักเรียนแต่ละคน:
   - สร้าง `masterToken` จาก student_id + national_id
   - สร้าง `compositeToken` จาก masterToken + gradeLevel + classroom + academicYear + subjectCode
   - เข้ารหัส `real_name` ด้วย `aesGcmEncrypt(name, cryptoKey)`
   - เข้ารหัส `student_id` ด้วย `aesGcmEncrypt(studentId, cryptoKey)`
5. **Bulk Upload**: ส่งเฉพาะ `{anonymized_id, encrypted_name, encrypted_student_id, classroom_id, encryption_key_version}` ไปยัง Server
6. **Local Mapping**: บันทึก Mapping ระหว่าง compositeToken และ realName ไว้ในเครื่อง Client เท่านั้น

PAGE 3: DATABASE ARCHITECTURE & OPTIMIZATION

3.1 ภาพรวม Database Architecture

ระบบใช้ **SQLAlchemy ORM** กับ **PostgreSQL** เป็น Primary Database และรองรับ **SQLite** สำหรับ Development Environment โดยใช้ Async Engine ทั้งหมด

3.2 Table Groups (8 กลุ่มหลัก)

กลุ่มที่ 1: Core SRO Engine (5 ตาราง)

Table	หน้าที่	Key Indexes
`documents`	เก็บเอกสารต้นฉบับและ Extracted Text	`id` , `title` , `created_at`
`text_chunks`	เก็บ Text Segments จากการ Chunking	`document_id` , `processing_status` , `chunk_type`
`sros`	เก็บ Scientific Relationship Object Triples	`subject_entity_id` , `object_entity_id` , `confidence`
`entities`	เก็น Standardized Entities (Nodes)	`entity_name` , `canonical_name` , `entity_type` , `is_curated`
`knowledge_graph`	เก็บ Graph Edges ระหว่าง Entities	`sro_id` , `subject_entity_id` , `object_entity_id` , `weight` , `confidence`

กลุ่มที่ 2: KRU Educational Domain (12+ ตาราง)

Table	หน้าที่
`schools`	ข้อมูลโรงเรียน
`teachers`	ข้อมูลครู (เชื่อมกับ users)
`classrooms`	ข้อมูลห้องเรียน (grade_level, semester, academic_year)
`classroom_subjects`	Bridge table: ห้องเรียน ↔ วิชา ↔ ปีหลักสูตร
`academic_contexts`	บริบททางวิชาการ
`lesson_plans`	แผนการสอน
`learning_objectives`	จุดประสงค์การเรียนรู้ (Bloom's Taxonomy)
`assessment_items`	ข้อสอบ/แบบประเมิน
`students`	ข้อมูลนักเรียน (เข้ารหัส)

Table	หน้าที่
`student_performances`	ผลการเรียน (ใช้ Anonymized ID)
`curriculum_standards`	มาตรฐานหลักสูตร OBEC
`kru_assessment_maps`	การ映射ครู ↔ ข้อสอบ ↔ ห้องเรียน

กลุ่มที่ 3: Encryption & Security (4 ตาราง)

Table	หน้าที่
`teacher_encryption_keys`	Key ของครูพร้อม Versioning
`school_encryption_keys`	Key ของโรงเรียนพร้อม Versioning
`teacher_consents`	บันทึก PDPA Consent
`key_migration_logs`	Audit Trail การเปลี่ยน Key

กลุ่มที่ 4: Taxonomy Cache (1 ตาราง)

Table	หน้าที่	Key Feature
`taxonomy_cache`	Cache ผลวิเคราะห์ Bloom/OBEC	SHA-256 Hash ของ Question Body เป็น Unique Key ป้องกันการวิเคราะห์ซ้ำ

กลุ่มที่ 5: LLM Usage Telemetry (1 ตาราง)

Table	หน้าที่	Key Fields
`llm_usage_log`	บันทึก LLM Usage สำหรับ Billing	`provider` , `model` , `task_type` , `prompt_tokens` , `output_tokens` , `estimated_cost_usd` , `estimated_cost_thb` , `latency_ms`

กลุ่มที่ 6: Exam Processing (1 ตาราง)

Table	หน้าที่	Key Fields
`exam_batches`	ติดตามการสกัดข้อสอบ	`source_filename` , `status` , `pipeline_version` , `layout_type` , `total_items_extracted` , `raw_llm_response`

กลุ่มที่ 7: SRO Relationships (KRU Plugin) (1 ตาราง)

Table	หน้าที่
`kru_sro_relationships`	เก็บ SRO Triples เฉพาะของ KRU Plugin

กลุ่มที่ 8: System & Metadata (2+ ตาราง)

Table	หน้าที่
`users`	ผู้ใช้ระบบ (Authentication)
`projects`	โปรเจกต์ (Workspace)
`extraction_runs`	การรัน SRO Extraction

3.3 Connection Pool Configuration

```
engine = create_async_engine(
    settings.database_url,
    pool_size=settings.db_pool_size,        # จำนวน Connection หลัก
    max_overflow=settings.db_max_overflow,  # Connection เพิ่มเมื่อมี Load สูง
    pool_timeout=settings.db_pool_timeout,  # Timeout รอ Connection
    pool_recycle=settings.db_pool_recycle,  # รีเซ็ต Connection เก่า
    echo=settings.debug,                    # SQL Logging (Dev only)
    future=True,
)
```

3.4 Indexing Strategy

ระบบออกแบบ Index แบบ **Multi-Column Composite Index** สำหรับ Query ที่ใช้บ่อย:

ตัวอย่างจาก `text_chunks` :

- `idx_chunk_document_status` : (document_id, processing_status) — สำหรับ Query สถานะการประมวลผลตามเอกสาร
- `idx_chunk_document_type` : (document_id, chunk_type) — สำหรับกรอง Chunk ตามประเภท
- `idx_chunk_document_section` : (document_id, section) — สำหรับค้นหาตาม Section
- `uq_document_chunk_index` : (document_id, chunk_index) — Unique Constraint ป้องกัน Chunk ซ้ำ

ตัวอย่างจาก `knowledge_graph` :

- Index บน `sro_id` , `subject_entity_id` , `object_entity_id` , `weight` , `confidence` , `direction` , `graph_type` , `is_validated`

3.5 Data Integrity Constraints

ระบบใช้ **Check Constraints** และ **Unique Constraints** อย่างครอบคลุม:

```
-- text_chunks constraints
CHECK (chunk_index ⩾ 0)
CHECK (word_count ⩾ 0)
CHECK (character_count ⩾ 0)
CHECK (start_page ⩾ 0)
CHECK (end_page ⩾ 0)
UNIQUE (document_id, chunk_index)

-- kru_assessment_maps constraints
UNIQUE (teacher_id, exam_item_id, usage_type)
```

3.6 Caching Strategy

3.6.1 Taxonomy Cache (Database-Level)

`taxonomy_cache` ใช้ **SHA-256 Hash** ของ Question Body เป็น Unique Key เพื่อ:

- ป้องกันการเรียก LLM ซ้ำสำหรับคำถามเดียวกัน
- ลด Token Cost และ Latency
- รองรับ Cache Invalidation ผ่าน `updated_at` Timestamp

3.6.2 SQLite WAL Mode (Development)

สำหรับ Development Environment ที่ใช้ SQLite:

- `PRAGMA journal_mode=WAL` — Write-Ahead Logging สำหรับ Concurrent Read/Write
- `PRAGMA synchronous=NORMAL` — สมดุลระหว่าง Performance และ Durability
- `PRAGMA busy_timeout=3000` — รอ 3 วินาที เมื่อ Database Locked

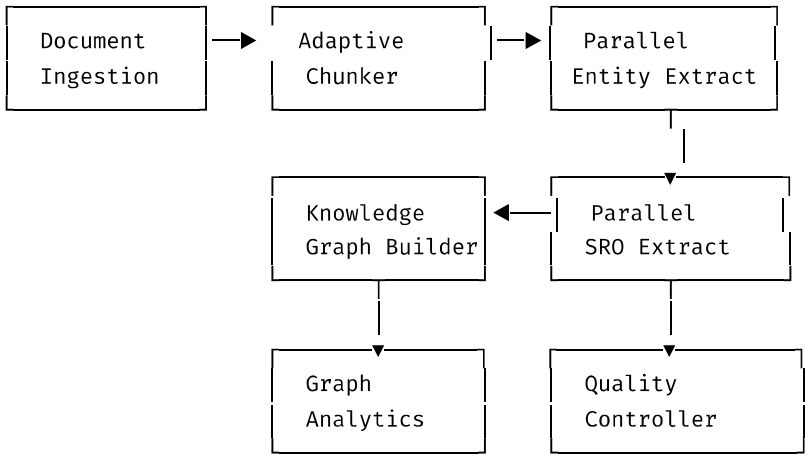
3.7 Denormalization & JSON Columns

ระบบใช้ **JSON/JSONB Columns** สำหรับ Flexible Metadata แทนการสร้างตารางเพิ่ม:

Table	JSON Column	เก็บอะไร
`entities`	`aliases` , `synonyms` , `entity_metadata`	ชื่อแฝง, คำพ้องความ, Metadata เพิ่มเติม
`knowledge_graph`	`context` , `metadata`	บริบทความสัมพันธ์, Metadata ของ Edge
`text_chunks`	`chunk_metadata`	Content type, Chunk strategy, Document metadata
`exam_batches`	`raw_llm_response` , `filtered_telemetry`	LLM Response ดิบ, Telemetry ที่กรองแล้ว
`taxonomy_cache`	`obec_indicators`	ตัวชี้วัด OBEC แบบ JSON

PAGE 4: AI DOCUMENT PIPELINE & SRO KNOWLEDGE GRAPH ENGINE

4.1 ภาพรวม AI Document Pipeline



4.2 Adaptive Chunking Engine

`AdaptiveChunker` ทำการแบ่งเอกสารอัตโนมัติโดย **ตรวจจับประเภทเอกสาร** และปรับ Chunking Strategy ตาม:

4.2.1 Content Type Detection (6 ประเภท)

Content Type	Min Chunk	Max Chunk	Overlap Ratio	Section Patterns
Academic Paper	800 chars	3,000 chars	0.15	Abstract, Introduction, Methods, Results, Discussion
Medical Report	600 chars	2,500 chars	0.20	Patient Info, Clinical Findings, Diagnosis, Treatment
Legal Document	1,000 chars	4,000 chars	0.10	WHEREAS, Article, Section, Clause
News Article	400 chars	2,000 chars	0.10	By-line, Dateline, Section breaks
Technical Manual	500 chars	3,000 chars	0.15	Chapter, Section, Step, WARNING, CAUTION
General Text	500 chars	4,000 chars	0.10	Paragraph breaks, Numbered lists

4.2.2 Semantic Boundary Detection

ระบบตรวจจับ **Semantic Boundaries** หลายระดับ:

1. **Section Patterns**: Regex ตามประเภทเอกสาร (เช่น `\nAbstract\b` , `\nMethods\b`)
2. **General Semantic Patterns**: Double newlines, Sentence boundaries, Numbered lists, Headers, Horizontal rules, Figure/Table references
3. **Sentence Boundaries**: ใช้ NLTK `sent_tokenize()` เป็น Fallback
4. **Minimum Distance Filter**: กรอง Boundary ที่ใกล้กันเกินไป (< 50 characters)

4.2.3 Dynamic Overlap Calculation

ระบบคำนวณ **Dynamic Overlap** ระหว่าง Chunks โดยใช้ **Jaccard Similarity**:

- Similarity > 0.8 → Overlap ลด 50% (ลด Redundancy)
- Similarity > 0.6 → Overlap ลด 30%
- Similarity < 0.6 → Overlap เต็ม (รักษา Context)

4.2.4 Semantic Score Calculation

แต่ละ Chunk ได้รับ **Semantic Score** (0.0–1.0) คำนวณจาก:

Factor	Weight	รายละเอียด
Length Score	0.30	ชอบ Chunk ขนาดกลาง (~1,500 chars)
Sentence Completeness	0.30	จบประโยคสมบูรณ์
Word Diversity	0.20	ความหลากหลายของคำ (Unique/Total)
Entity Density	0.20	ความหนาแน่นของ Entity (Placeholder)

4.3 Parallel Entity & SRO Extraction

4.3.1 Batch Processing

ระบบแบ่ง Chunks เป็น **Batches** (default: 10 chunks/batch) และประมวลผลแต่ละ Batch แบบขนานโดยใช้ `asyncio.gather()` :

```
batch_tasks = [
    self._extract_entities_from_chunk(chunk)
    for chunk in batch
]
batch_results = await asyncio.gather(*batch_tasks, return_exceptions=True)
```

4.3.2 Retry with Exponential Backoff

Parameter	ค่า	รายละเอียด
Max Retries	3	จำนวนครั้ง Retry สูงสุด
Base Delay	1.0 วินาที	Delay เริ่มต้น
Exponential Backoff	`delay = base * (2 ^ attempt)`	1s → 2s → 4s → 8s

4.3.3 ThreadPoolExecutor

ใช้ `ThreadPoolExecutor(max_workers=4)` สำหรับรัน Synchronous Callback ใน Async Context โดยไม่ Block Event Loop

4.4 Quality Control Engine

`QualityController` ประเมินคุณภาพ SRO และ Entity ด้วย **5 Metrics**:

Metric	Weight	High	Medium	Low	Reject
Confidence	0.30	≥ 0.9	≥ 0.7	≥ 0.5	< 0.3
Completeness	0.25	≥ 0.9	≥ 0.7	≥ 0.5	< 0.3
Consistency	0.20	≥ 0.8	≥ 0.6	≥ 0.4	< 0.2
Validity	0.15	≥ 0.9	≥ 0.7	≥ 0.5	< 0.3
Relevance	0.10	≥ 0.8	≥ 0.6	≥ 0.4	< 0.2

Overall Score = Σ(metric × weight)

Strict Mode: เพิ่ม Threshold ทุกระดับ +0.1

4.4.1 Consistency Checks

- **Duplicate Detection**: เปรียบเทียบ (subject, relationship, object) แบบ Case-insensitive
- **Contradictory Detection**: ตรวจจับความขัดแย้ง เช่น `causes` vs `prevents`, `enables` vs `inhibits`
- **Invalid Pattern Matching**: กรองความสัมพันธ์ที่ไม่สมบูรณ์ เช่น "is a ...", "very ...", "good ..."

4.5 Knowledge Graph Construction

4.5.1 Graph Data Structure

ใช้ **NetworkX DiGraph** สำหรับ In-Memory Graph Construction:

- **Nodes**: `GraphNode` (entity_id, canonical_name, entity_type, aliases, properties, confidence)
- **Edges**: `GraphEdge` (source_id, target_id, relationship_type, weight, confidence, metadata)

4.5.2 Entity Standardization

`EntityStandardizer` ทำการ Standardize Entity Names ตามประเภท:

- **Person**: Title Case, ลด Stopwords, Pattern Matching (Dr., Prof., Mr.)
- **Organization**: Title Case, Pattern Matching (University, College, Inc.)
- **Location**: Title Case, Pattern Matching (City, State, Country)
- **Direct Mappings**: เช่น "harvard" → "Harvard University", "who" → "World Health Organization"

4.5.3 Edge Weight Calculation

`WeightCalculator` คำนวณ Edge Weight โดย:

```
final_weight = base_weight × avg_confidence × evidence_multiplier × context_strength
```

Base Weights ตาม Relationship Type:

Relationship	Base Weight
CAUSES	0.9
ENABLES	0.8
CONTRADICTS	0.8
INFLUENCES	0.7
SUPPORTS	0.7
REQUIRES	0.6
PRECEDES	0.6
FOLLOWS	0.6
PART_OF	0.5
RELATED_TO	0.3

Evidence Multiplier: `min(1.0, 0.7 + 0.3 × ln(evidence_count + 1))` — Diminishing Returns

4.5.4 Graph Consolidation

หลังจากสร้าง Graph แล้ว ระบบทำ Consolidation 3 ขั้นตอน:

1. **Merge Duplicate Edges**: รวม Edge ระหว่าง Node คู่เดียวกัน เก็บ Edge ที่มี Weight สูงสุด และ Aggregate Evidence Count
2. **Update Centralities**: คำนวณ Degree Centrality, Betweenness Centrality, และ Closeness Centrality สำหรับทุก Node
3. **Filter Low-Quality Elements**: ลบ Edge ที่ Weight < 0.3 และ Node ที่ Confidence < 0.5 หรือ Isolated Nodes

4.5.5 Graph Metrics

ระบบคำนวณ Graph Metrics สำหรับ Analytics:

- **Density**: ความหนาแน่นของ Graph
- **Average Clustering Coefficient**: ความโดดเด่นของ Clustering
- **Average Path Length** และ **Diameter** (สำหรับ Connected Graph)
- **Degree Distribution**: Average, Max, Min Degree

4.6 LLM Token Cost Tracking

`LLMUsageLog` บันทึก Token Usage ทุกครั้งที่เรียก LLM สำหรับ Billing และ Monitoring:

Field	รายละเอียด
`provider`	LLM Provider (เช่น OpenAI, Anthropic)
`model`	ชื่อ Model ที่ใช้
`task_type`	ประเภท Task (เช่น entity_extraction, sro_extraction)

Field	รายละเอียด
`prompt_tokens`	จำนวน Input Tokens
`output_tokens`	จำนวน Output Tokens
`estimated_cost_usd`	ค่าใช้จ่ายโดยประมาณ (USD) — ความละเอียด 6 ตำแหน่งทศนิยม
`estimated_cost_thb`	ค่าใช้จ่ายโดยประมาณ (THB) — ความละเอียด 4 ตำแหน่งทศนิยม
`latency_ms`	เวลาตอบสนองในมิลลิวินาที
`success`	สถานะสำเร็จ/ล้มเหลว

4.7 TextChunk Model (Persistent Storage)

`TextChunk` ใน Database เก็บข้อมูล Chunk อย่างละเอียด:

Field	Type	รายละเอียด
`chunk_index`	BigInteger	ลำดับ Chunk ในเอกสาร
`content`	Text	เนื้อหาของ Chunk
`word_count`	BigInteger	จำนวนคำ
`character_count`	BigInteger	จำนวนตัวอักษร
`section`	String(100)	Section ของเอกสาร
`chunk_type`	Enum	ABSTRACT, INTRODUCTION, METHODS, RESULTS, DISCUSSION, CONCLUSION, REFERENCES, APPENDIX, GENERAL
`processing_status`	Enum	CREATED, PROCESSING, COMPLETED, FAILED, PENDING
`start_page / end_page`	BigInteger	ตำแหน่งหน้า
`start_char / end_char`	BigInteger	ตำแหน่งตัวอักษร
`chunk_metadata`	JSON	Metadata เพิ่มเติม

4.8 Pipeline Progress Tracking

ระบบติดตาม Progress แบบ Real-time ผ่าน `PipelineProgress` :

- `current_stage` : ขั้นตอนปัจจุบัน (6 Stages)
- `total_chunks / processed_chunks / failed_chunks` : สถานะการประมวลผล
- `stage_progress` : Dict ของ Progress แต่ละ Stage (0.0–1.0)
- `estimated_completion` : เวลาที่คาดว่าจะเสร็จ
- `errors / warnings` : รายการ Error และ Warning

Progress ถูกส่งกลับผ่าน **Progress Callback** ซึ่งรันใน ThreadPool เพื่อหลีกเลี่ยงการ Block Event Loop

> **หมายเหตุ**: เอกสารนี้จัดทำจากการวิเคราะห์ Source Code โดยตรง (Source: `backend/` และ `frontend/` directories) ข้อมูลทั้งหมดอ้างอิงจาก Code จริงในระบบ SRO App V2.8 ไม่มีการสร้างข้อมูลขึ้นมาเอง (No Fabricated Data)