Adaptation Flow: How It Actually Works

Date: 2026-01-16 **Total LOC in Adaptation System:** ~2,700 **Key Files:** `adaptation_engine.py`, `gemini_client.py`, `prompts.py`, `sharder.py`

Table of Contents

---

1. High-Level Flow

```
INPUT: Source JSON + Target Scenario Prompt
              |
              v
    +-------------------+
    | FACTSHEET         |  <-- 1 LLM call (Gemini)
    | EXTRACTION        |      Extracts: company, manager, poison list, KLOs
    +-------------------+
              |
              v
    +-------------------+
    | SHARDER           |  <-- Pure Python (no LLM)
    |                   |      Splits JSON into 15 shards
    +-------------------+
              |
              v
    +-------------------+
    | PARALLEL          |  <-- 13 LLM calls (Gemini) in parallel
    | ADAPTATION        |      Each shard adapted with factsheet context
    +-------------------+
              |
              v
    +-------------------+
    | VERIFICATION      |  <-- Code-based verification (no LLM)
    | LOOP              |      Checks placeholders, poison terms, formatting
    +-------------------+
              |
              v
    +-------------------+
    | MERGE & CLEANUP   |  <-- Pure Python
    |                   |      Combines shards, fixes URLs/placeholders
    +-------------------+
              |
              v
OUTPUT: Adapted JSON + Entity Mappings
```

---

2. Step 1: Factsheet Extraction

What It Does

Extracts global context BEFORE any adaptation happens. This ensures all 13 shards use the same company name, manager name, KLOs, and poison list.

File Location

- **Prompt:** `src/utils/prompts.py` lines 34-105
- **Execution:** `src/utils/gemini_client.py` lines 352-439

The Prompt (FACTSHEET_PROMPT)

```
You MUST extract a complete factsheet. ALL fields are REQUIRED - do not skip any.

## SOURCE SCENARIO (what we're adapting FROM - extract poison terms from here):
{source_scenario}

## TARGET SCENARIO (what we're adapting TO - extract company/role/context from here):
{target_scenario}

[BLOCK] CRITICAL: You MUST fill ALL fields below. Empty fields = REJECTED.

## POISON LIST INSTRUCTIONS (VERY IMPORTANT):
The poison_list must contain ALL terms from the SOURCE scenario that should NOT appear in the adapted content
You MUST extract:
1. ALL person names from SOURCE (both first name AND full name separately)
2. ALL company/organization names from SOURCE
3. ALL industry-specific jargon from SOURCE that doesn't fit TARGET
4. ALL role-specific terms from SOURCE (e.g., if SOURCE is about hiring: "candidate", "interview", "recruitme

Example: If SOURCE has manager "Elizabeth Carter" at "Summit Innovations" doing "hiring":
poison_list: ["Elizabeth", "Elizabeth Carter", "Summit Innovations", "hiring", "candidate", "interview", "rec

Return this EXACT JSON structure with ALL fields populated:

{
  "company": {
    "name": "Exact company name from TARGET scenario",
    "industry": "Industry type from TARGET"
  },
  "products": {
    "main_product": "Primary product/service from TARGET",
    "product_details": ["3-5 specific attributes"]
  },
  "context": {
    "challenge": "Main business challenge from TARGET",
    "market": "Market context with specifics"
  },
  "learner_role": {
    "role": "Job title the learner plays in TARGET",
    "key_responsibilities": ["3-4 tasks they must do"]
  },
  "reporting_manager": {
    "name": "Create a realistic manager name for TARGET industry",
    "role": "Appropriate manager title",
    "email": "firstname.lastname@companyname.com",
    "gender": "Male or Female"
  },
```

```
  "industry_context": {
    "kpis": ["List 10-15 KPIs specific to TARGET industry"],
    "terminology": ["List 15-20 industry terms to use in TARGET"],
    "wrong_terms": ["Terms from SOURCE industry that don't fit TARGET"]
  },
  "klos": [
    "KLO 1: Specific skill/capability learner demonstrates in TARGET context",
    "KLO 2: Second learning outcome aligned to TARGET challenge",
    "KLO 3: Third learning outcome for TARGET scenario"
  ],
  "poison_list": ["MINIMUM 15 terms from SOURCE"],
  "citable_facts": ["10-15 specific facts/numbers relevant to TARGET industry"]
}

[BLOCK] VALIDATION CHECKLIST:
- [ ] company.name is NOT empty
- [ ] reporting_manager.name is a real name (First Last format)
- [ ] reporting_manager.email matches the company domain
- [ ] klos has EXACTLY 3 items
- [ ] poison_list has AT LEAST 15 terms extracted from SOURCE
- [ ] industry_context.kpis has AT LEAST 10 items

Return ONLY the JSON. No explanations.
```

How It's Called

```python
# File: gemini_client.py, lines 352-439
@traceable
async def extract_global_factsheet(
    source_scenario: str,
    target_scenario: str,
) -> dict[str, Any]:

    # Truncate scenarios to 2000 chars
    safe_source = safe_str(source_scenario)[:2000]
    safe_target = safe_str(target_scenario)[:2000]

    # Build prompt from prompts.py
    prompt = build_factsheet_prompt(safe_source, safe_target)

    # Call Gemini with temperature=0.2 for consistency
    result = await call_gemini(prompt, temperature=0.2, max_tokens=8192)

    # Filter poison list to remove common English words
    filtered_poison_list = filter_poison_list(raw_poison_list)

    # Augment with names extracted via regex from source
    augmented_poison_list = augment_poison_list_with_names(filtered_poison_list, source_scenario)

    return result
```

Output Example

```
{
  "company": {
    "name": "EcoChic Threads",
    "industry": "Sustainable Fashion Retail"
  },
  "reporting_manager": {
```

```
    "name": "Maya Sharma",
    "role": "Director of Market Strategy",
    "email": "maya.sharma@ecochic.com",
    "gender": "female"
  },
  "poison_list": [
    "Elizabeth", "Elizabeth Carter", "Summit Innovations",
    "HR", "hiring", "candidate", "interview", "recruitment"
  ],
  "klos": [
    "Analyze market opportunity using TAM/SAM/SOM",
    "Evaluate internal capabilities and resource fit",
    "Develop go/no-go recommendation with risk assessment"
  ],
  "industry_context": {
    "kpis": ["CAC", "LTV", "Gross Margin", "Market Share", "Conversion Rate"],
    "terminology": ["market entry", "competitive analysis", "PESTEL"],
    "wrong_terms": ["HR metrics", "hiring KPIs", "recruitment funnel"]
  }
}
```

Problem Identified

The poison list misses domain-IMPLIED terms like "HR" because the SOURCE scenario text may say "Elizabeth Carter manages talent acquisition" without literally using "HR". The regex-based augmentation only extracts NAMES, not domain vocabulary.

---

3. Step 2: Shard Splitting

What It Does

Splits the simulation JSON into 15 semantically independent shards for parallel processing.

File Location

- **Code:** `src/stages/sharder.py` lines 50-150

Shard Definitions

```
SHARD_DEFINITIONS = [
    # LOCKED - Never sent to LLM
    "workspace_ids",        # System IDs
    "scenario_options",     # Original scenarios list

    # UNLOCKED - Sent to LLM for adaptation
    "lesson_information",   # Contains manager emails, company info
    "overview",             # Scenario overview text
    "guidelines",           # Student guidelines
    "simulation_flow",      # Core activities + submission questions + resources
    "submission_questions", # Top-level assessment questions (often empty)
    "rubric",               # Grading criteria
    "assessment_criterion", # KLO definitions
    "emails",               # Manager communications
    "characters",           # Personas
    "resources",            # Learning materials
    "workplace_scenario",   # Scenario narrative
    "activities",           # Task definitions
    "stage_definitions",    # Workflow stages
]
```

Lock States

```python
class LockState(Enum):
    UNLOCKED = "unlocked"           # Can be adapted
    LOCKED = "locked"               # Never sent to LLM
    LOCKED_AFTER_FIX = "locked_after_fix"  # Locked after structural fix
```

How Sharding Works

```python
# File: sharder.py
def shard(self, input_json: dict) -> list[Shard]:
    topic_data = input_json.get("topicWizardData", {})
    shards = []

    for shard_def in SHARD_DEFINITIONS:
        shard_key = self._get_shard_key(shard_def)
        content = topic_data.get(shard_key, {})

        lock_state = LockState.LOCKED if shard_def in LOCKED_SHARDS else LockState.UNLOCKED

        shard = Shard(
            id=shard_def,
            name=shard_def.replace("_", " ").title(),
            content=content,
            lock_state=lock_state,
            hash=compute_hash(content)
        )
        shards.append(shard)

    return shards
```

Problem Identified

KLOs are in `assessment_criterion` shard, but submission questions are in `simulation_flow` shard. When adapted independently, there's no cross-shard awareness for KLO-question alignment.

---

4. Step 3: Parallel Adaptation

What It Does

Sends each unlocked shard to Gemini with the factsheet context injected.

File Location

- **Prompt Builder:** `src/utils/prompts.py` lines 314-430
- **Execution:** `src/utils/gemini_client.py` lines 446-499

The Prompt Structure (Per Shard)

The prompt is assembled in this order:

```
1. HARD_BLOCKS_WITH_CONSEQUENCES
   - Regex patterns that will trigger rejection
   - Poison term list
   - Formatting rules

2. DYNAMIC_RULES (from rules.py)
   - Shard-specific validation rules

3. KLO_ALIGNMENT (if relevant shard)
```

```
   - KLOs from factsheet for alignment

4. MANDATORY_VERIFICATION_FIELDS
   - JSON structure LLM must include

5. CONTEXT_SECTION
   - Company name, manager, industry
   - Source -> Target transformation

6. SHARD_GUIDANCE
   - Shard-specific rules (resources = 500+ words, etc.)

7. CONTENT_TO_ADAPT
   - Actual JSON shard content

8. OUTPUT_FORMAT
   - Expected JSON response structure
```

Complete Shard Adaptation Prompt

```
## [BLOCK] HARD BLOCKS - VIOLATIONS = AUTOMATIC REJECTION

Your output will be programmatically scanned. These checks CANNOT be bypassed:

### BLOCK 1: ZERO PLACEHOLDERS
**Regex applied to your output:** `\[[^\]]+\]`
This catches: [anything in brackets], [X], [TBD], [manager name], etc.

If regex finds ANY match --> OUTPUT REJECTED, you must regenerate.

Examples that WILL trigger rejection:
- "growing at [X]%" --> REJECTED
- "[industry-specific metric]" --> REJECTED
- "Contact [manager name]" --> REJECTED

[OK] Replace ALL brackets with actual values before outputting.

### BLOCK 2: ZERO POISON TERMS
**String search applied for each term:** ["Elizabeth", "Summit Innovations", "HR", "hiring", ...]

If ANY term found (case-insensitive) --> OUTPUT REJECTED.

### BLOCK 3: SENDER CONSISTENCY
**Cross-reference check applied:**
- Manager name in factsheet: Maya Sharma
- All email "from" fields must contain: Maya Sharma
- All email signatures must contain: Maya Sharma

If mismatch detected --> OUTPUT REJECTED.

### BLOCK 4: CLEAN FORMATTING
**Regex applied:** `\.(png|jpg|com|org)\.`
Catches trailing periods on emails, URLs, filenames.

- WRONG: "sophia.chen@retail.com." --> REJECTED
- RIGHT: "sophia.chen@retail.com"

### BLOCK 5: COMPLETE CONTENT
**Scan applied for:** `...`, sentences ending without punctuation, `TBD`, `TODO`
```

```
Incomplete content --> OUTPUT REJECTED.

---

## [TARGET] RULES FOR THIS SHARD (from rules.py)
[Dynamic rules inserted here based on shard type]

## [TARGET] KLOs - ALL CONTENT MUST ALIGN TO THESE:
  KLO1: Analyze market opportunity using TAM/SAM/SOM
  KLO2: Evaluate internal capabilities and resource fit
  KLO3: Develop go/no-go recommendation with risk assessment

[BLOCK] Activities, questions, and resources MUST directly support these KLOs.
[BLOCK] Do NOT include content about unrelated topics (e.g., HR hiring if KLOs are about market analysis).

---

## [CHECK] MANDATORY VERIFICATION OUTPUT

Your output MUST include these verification fields:

```json
{
  "adapted_content": { ... },

  "poison_scan_proof": {
    "terms_i_searched_for": [/* MUST list at least 15 terms from poison list */],
    "terms_found_in_my_output": [],  // MUST be empty array
    "sections_i_checked": ["lessonInformation", "emails", "resources", "guidelines"]
  },

  "placeholder_scan_proof": {
    "patterns_i_searched_for": ["[", "{{", "TBD", "TODO", "XXX"],
    "brackets_found": 0,  // MUST be 0
    "replacements_i_made": [/* list any [X] you replaced with real values */]
  },

  "sender_consistency_proof": {
    "manager_from_factsheet": "Maya Sharma",
    "manager_in_email_from_fields": "Maya Sharma",
    "manager_in_signatures": "Maya Sharma",
    "all_match": true
  },

  "formatting_proof": {
    "trailing_dots_found": 0,
    "incomplete_sentences_found": 0
  }
}
```

ADAPTATION CONTEXT

Transform from: HR Hiring/Selection at Summit Innovations... **Transform to:** Market Entry Analysis for EcoChic Threads...
```

[BLOCK] CRITICAL: ENTITY SEPARATION (DO NOT MIX!)

- **COMPANY NAME:** EcoChic Threads <– Use this EXACTLY for company references
- **MANAGER NAME:** Maya Sharma <– Use this EXACTLY for manager/person references

Target Company: EcoChic Threads (Sustainable Fashion Retail) **Learner Role:** Junior Business Analyst **Manager:** Maya Sharma (Director of Market Strategy) - maya.sharma@ecochic.com

Use these industry terms: market entry, competitive analysis, PESTEL, TAM/SAM **Use these KPIs:** CAC, LTV, Gross Margin, Market Share

---

SHARD: Resources

[WARN] CRITICAL: Resources MUST be COMPREHENSIVE and LEARNER-READY

CONTENT REQUIREMENTS (NON-NEGOTIABLE):

1. **MINIMUM LENGTH**: Each resource markdownText MUST be 500+ words
2. **STRUCTURE**: Use proper markdown headings (##, ###) and bullet points
3. **STATISTICS**: Every stat needs source citation: "Value (Source: Organization Year)"
   - Market size: "$X.X billion (Source: IBISWorld 2024)"
   - Growth rate: "X.X% CAGR (Source: Grand View Research 2024)"
4. **COMPETITOR ANALYSIS**: List 3-5 real competitors
5. **FRAMEWORKS**: Include SWOT, PESTEL, financial metrics
6. **NO EMPTY SECTIONS**: Every field must have substantial content
   - No "[placeholder]", no "TBD", no truncated sentences

[BLOCK] REJECTION TRIGGERS: - markdownText under 300 words –> REJECTED - Statistics without sources –> REJECTED - Empty or placeholder content –> REJECTED

CONTENT TO ADAPT:

```
{
  "resources": [
    {
      "id": "resource_1",
      "title": "Hiring Process Overview",
      "markdownText": "... original content ..."
    }
  ]
}
```

OUTPUT FORMAT

Return this EXACT JSON structure:

```
{
  "adapted_content": {
    /* transformed content - same keys as input */
  },

  "entity_mappings": {
    "old_term": "new_term"
  },
```

```
  "poison_scan_proof": { ... },
  "placeholder_scan_proof": { ... },
  "sender_consistency_proof": { ... },
  "formatting_proof": { ... }
}
```

CRITICAL: - Preserve all “id” and “Id” fields exactly - Use “EcoChic Threads” explicitly (not “the company”) - Every statistic needs a source citation

Return ONLY valid JSON. No markdown fences. No explanations outside JSON.

How It's Called

```python
# File: gemini_client.py, lines 446-499
@traceable
async def adapt_shard_content(
    shard_id: str,
    shard_name: str,
    content: dict,
    source_scenario: str,
    target_scenario: str,
    global_factsheet: dict = None,
    rag_context: str = "",
    max_verification_retries: int = 2,
) -> tuple[dict, dict]:

    # Build prompt using prompts.py
    prompt = build_shard_adaptation_prompt(
        shard_id=shard_id,
        shard_name=shard_name,
        content=content,
        source_scenario=safe_source,
        target_scenario=safe_target,
        global_factsheet=factsheet,
        rag_context=safe_rag,
    )

    # Call Gemini
    result = await call_gemini(prompt, temperature=0.3)

    # Extract adapted content
    adapted_content = extract_adapted_content(result)
    entity_mappings = result.get("entity_mappings", {})

    return (adapted_content, entity_mappings)
```

Parallel Execution

```python
# File: adaptation_engine.py
async def adapt(self, input_json: dict, scenario_prompt: str):
    # 1. Extract factsheet (1 LLM call)
    global_factsheet = await extract_global_factsheet(source, target)

    # 2. Shard the JSON
    shards = Sharder().shard(input_json)

    # 3. Adapt shards in PARALLEL
    unlocked_shards = [s for s in shards if s.lock_state == LockState.UNLOCKED]
```

```
    tasks = [
        adapt_shard_content(
            shard_id=shard.id,
            shard_name=shard.name,
            content=shard.content,
            source_scenario=source,
            target_scenario=target,
            global_factsheet=global_factsheet,
        )
        for shard in unlocked_shards
    ]

    # Run all 13 shards in parallel
    results = await asyncio.gather(*tasks, return_exceptions=True)
```

---

5. Step 4: Verification Loop

What It Does

After LLM generates content, code-based verification checks for issues. If issues found, regenerates with feedback.

File Location

- **Verification Code:** `src/utils/prompts.py` lines 500-700

Verification Function

```
def strict_verify_output(
    adapted_content: dict,
    factsheet: dict,
    shard_id: str,
) -> VerificationResult:
    """
    Hard code verification of LLM output.

    Checks:
    1. Placeholder patterns ([X], {{Y}}, TBD, TODO)
    2. Poison terms (case-insensitive search)
    3. Manager name consistency
    4. Trailing dots on URLs/emails
    5. Incomplete sentences
    """
    issues = []

    content_str = json.dumps(adapted_content)

    # === CHECK 1: PLACEHOLDERS ===
    placeholder_patterns = [
        r'\[[^\]]+\]',           # [anything]
        r'\{\{[^}]+\}\}',        # {{anything}}
        r'\bTBD\b',              # TBD
        r'\bTODO\b',             # TODO
        r'\bXXX\b',              # XXX
        r'\[X\]',                # [X]
    ]

    for pattern in placeholder_patterns:
```

```
        matches = re.findall(pattern, content_str, re.IGNORECASE)
        if matches:
            issues.append(f"PLACEHOLDER: Found {matches[:3]}")

    # === CHECK 2: POISON TERMS ===
    poison_list = factsheet.get("poison_list", [])
    for term in poison_list:
        if term.lower() in content_str.lower():
            issues.append(f"POISON: Found '{term}'")

    # === CHECK 3: MANAGER CONSISTENCY ===
    expected_manager = factsheet.get("reporting_manager", {}).get("name", "")
    # Check email from fields, signatures, etc.

    # === CHECK 4: TRAILING DOTS ===
    trailing_dot_patterns = [
        r'\.com\.',
        r'\.org\.',
        r'\.png\.',
        r'\.jpg\.',
    ]
    for pattern in trailing_dot_patterns:
        if re.search(pattern, content_str):
            issues.append(f"TRAILING DOT: {pattern}")

    return VerificationResult(
        passed=len(issues) == 0,
        issues=issues,
    )
```

Regeneration on Failure

```
# File: gemini_client.py, lines 500-550
async def adapt_shard_content(...):
    for attempt in range(max_verification_retries):
        result = await call_gemini(prompt, temperature=0.3)
        adapted_content = extract_adapted_content(result)

        # Verify
        verification = strict_verify_output(adapted_content, factsheet, shard_id)

        if verification.passed:
            return (adapted_content, entity_mappings)

        # If failed, regenerate with feedback
        logger.warning(f"Verification failed (attempt {attempt+1}): {verification.issues}")

        # Build regeneration prompt with specific feedback
        prompt = build_regeneration_prompt(
            original_prompt=prompt,
            previous_output=adapted_content,
            issues=verification.issues,
        )

    # Return last attempt even if verification failed
    return (adapted_content, entity_mappings)
```

---

6. Step 5: Merge & Cleanup

What It Does

Combines the 15 adapted shards back into a single JSON and runs post-processing cleanup.

File Location

- **Merge:** `src/stages/sharder.py` lines 151-250
- **Cleanup:** `src/stages/sharder.py` lines 251-350

Merge Process

```python
def merge_shards(shards: list[Shard]) -> dict:
    """Merge adapted shards back into single JSON."""
    topic_data = {}

    for shard in shards:
        shard_key = _get_shard_key(shard.id)
        topic_data[shard_key] = shard.content

    return {"topicWizardData": topic_data}
```

Cleanup Process

```python
def cleanup_merged_json(topic_data: dict) -> dict:
    """Post-merge cleanup for known issues."""

    # 1. Fix truncated URLs
    from ..utils.content_fixer import fix_truncated_urls
    url_fixes = fix_truncated_urls(topic_data)

    # 2. Remove placeholders
    from ..utils.content_fixer import remove_placeholders
    placeholder_fixes = remove_placeholders(topic_data)

    # 3. Fix duplicate activity names
    from ..utils.content_fixer import fix_duplicate_activities
    duplicate_fixes = fix_duplicate_activities(topic_data)

    return topic_data
```

---

7. Complete Prompt Templates

Summary of All Prompts

Prompt	Location	Purpose	Tokens
FACTSHEET_PROMPT	prompts.py:34-105	Extract global context	~1,500
HARD_BLOCKS_WITH_CONSEQUENCES	prompts.py:107-173	Rejection rules	~800
MANDATORY_VERIFICATION_FIELDS	prompts.py:175-219	Force LLM to prove checking	~400
CONTEXT_SECTION	prompts.py:221-242	Company/manager context	~300
SHARD_GUIDANCE	prompts.py:244-249	Shard-specific rules	~200
OUTPUT_FORMAT	prompts.py:265-311	Expected JSON structure	~500
RESOURCE_RULES	prompts.py:437-473	500+ word requirement	~600

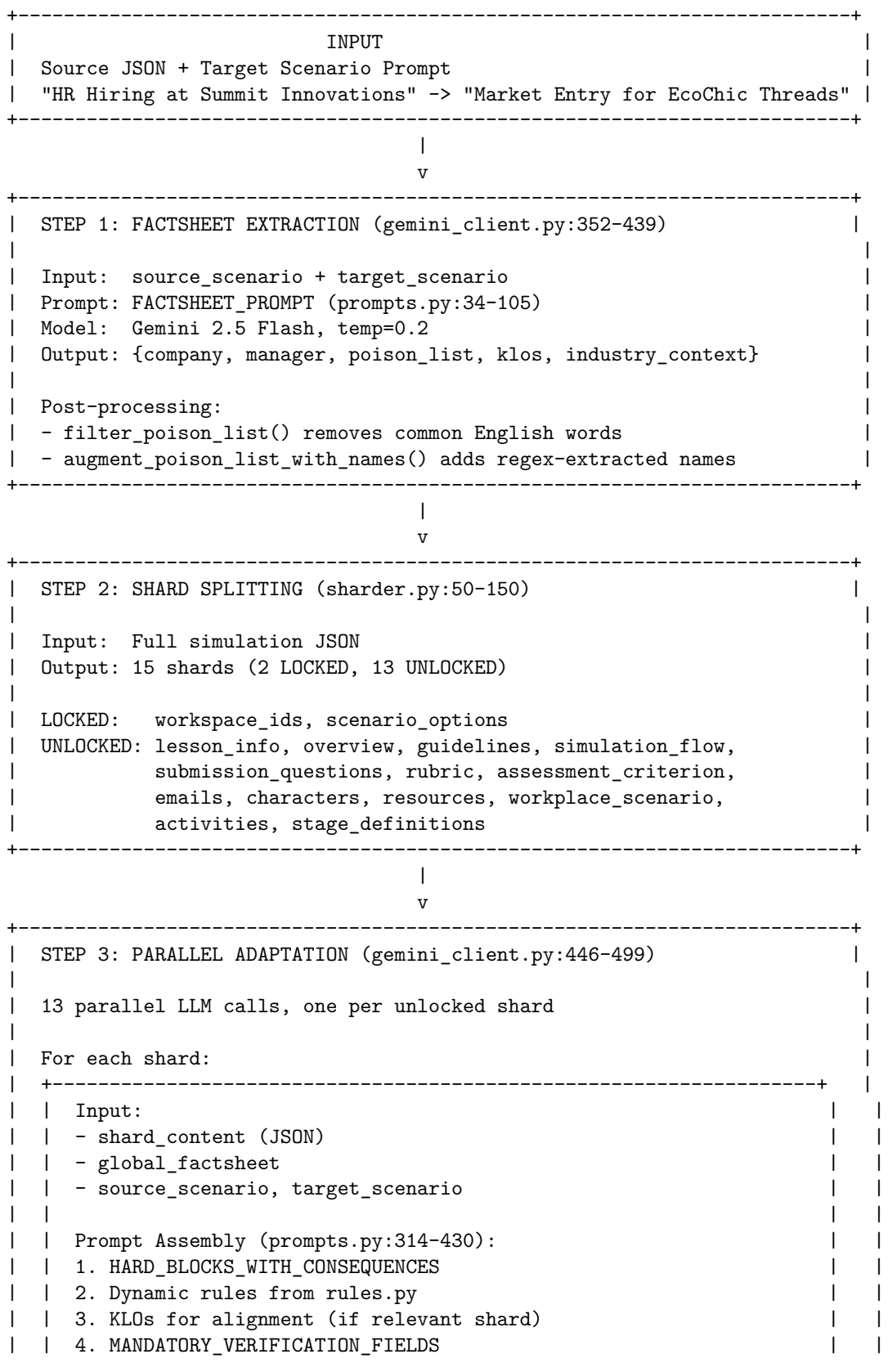
Total Prompt Size Per Shard

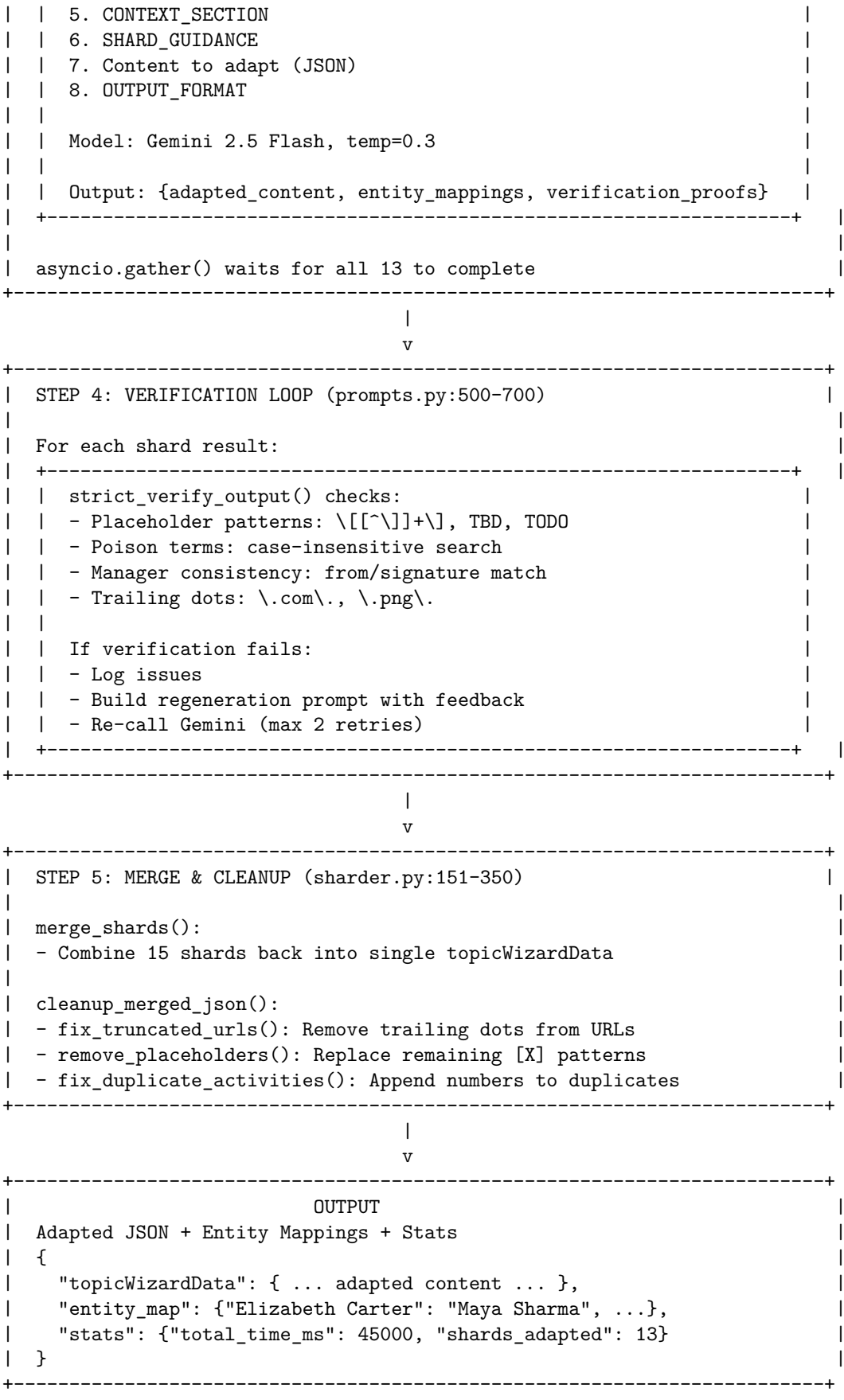
- Base prompt: ~3,500 tokens

- Content to adapt: Variable (500-5,000 tokens)
- **Total: ~4,000-8,500 tokens per shard**

---

8. Data Flow Diagram




5. CONTEXT_SECTION
6. SHARD_GUIDANCE
7. Content to adapt (JSON)
8. OUTPUT_FORMAT
Model: Gemini 2.5 Flash, temp=0.3
Output: {adapted_content, entity_mappings, verification_proofs}
asyncio.gather() waits for all 13 to complete
STEP 4: VERIFICATION LOOP (prompts.py:500-700)
For each shard result:
strict_verify_output() checks:
- Placeholder patterns: \[[^\]]+\], TBD, TODO
- Poison terms: case-insensitive search
- Manager consistency: from/signature match
- Trailing dots: \.com\., \.png\.
If verification fails:
- Log issues
- Build regeneration prompt with feedback
- Re-call Gemini (max 2 retries)
STEP 5: MERGE & CLEANUP (sharder.py:151-350)
merge_shards():
- Combine 15 shards back into single topicWizardData
cleanup_merged_json():
- fix_truncated_urls(): Remove trailing dots from URLs
- remove_placeholders(): Replace remaining [X] patterns
- fix_duplicate_activities(): Append numbers to duplicates
OUTPUT
Adapted JSON + Entity Mappings + Stats
{
"topicWizardData": { ... adapted content ... },
"entity_map": {"Elizabeth Carter": "Maya Sharma", ...},
"stats": {"total_time_ms": 45000, "shards_adapted": 13}
}

9. Critical Issues Identified

Issue 1: Poison List Misses Domain Terms

Location: `gemini_client.py` lines 129-168

Problem: The poison list only captures: - Person names (regex extraction) - Explicit terms in SOURCE scenario text

It MISSES domain-implied terms like "HR", "hiring", "candidate" because the SOURCE text may say "talent acquisition" without using "HR".

Impact: 185 "HR" occurrences leak through.

Issue 2: Cross-Shard KLO Blindness

Location: `prompts.py` lines 369-377

Problem: KLOs are injected into prompts for "simulation_flow", "resources", "rubrics" shards. But: - KLOs are generated in factsheet (not extracted from source) - Questions in simulation_flow don't see the actual `assessment_criterion` shard - No cross-reference between shards during adaptation

Impact: KLO-question alignment at 90% instead of 95%.

Issue 3: Resource Truncation

Location: `prompts.py` lines 437-473

Problem: Prompt says "MINIMUM 500+ words" but: - No actual word count verification - LLM may stop early if context window is tight - No retry specifically for short content

Impact: Resources truncated mid-PESTEL analysis.

Issue 4: Verification Loop Limited

Location: `gemini_client.py` lines 455-456

Problem: `max_verification_retries: int = 2` means only 2 regeneration attempts. If content fails verification twice, it's returned anyway.

Impact: Some issues slip through verification.

Issue 5: Entity Separation Warning Insufficient

Location: `prompts.py` lines 227-234

Problem: Prompt warns about entity separation but LLM still occasionally merges company and manager names in weird ways.

Impact: Occasional corrupted names like "EcoChic Maya Sharma Threads".

Summary

The adaptation system works through 5 main steps: 1. **Factsheet Extraction** - 1 LLM call to extract global context 2. **Shard Splitting** - Pure Python, 15 shards (2 locked, 13 unlocked) 3. **Parallel Adaptation** - 13 parallel LLM calls with factsheet injected 4. **Verification Loop** - Code-based checks with up to 2 retries 5. **Merge & Cleanup** - Combine shards and fix remaining issues

Key Metrics: - Total LLM calls: 14 (1 factsheet + 13 shards) - Prompt size per shard: 4,000-8,500 tokens - Model: Gemini 2.5 Flash - Temperature: 0.2 (factsheet), 0.3 (shards)

Critical Issues: 1. Poison list misses domain-implied terms 2. Cross-shard KLO alignment broken 3. Resource truncation not validated 4. Verification loop too limited 5. Entity separation occasionally fails

---

Analysis completed: 2026-01-16 Files analyzed: adaptation_engine.py, gemini_client.py, prompts.py, sharder.py Total LOC: ~2,700