Our next key task is to complete and harden the functionality for cleaning and removing hallucination, repetition, and unwanted artifacts from the final subtitle. Let's call it sanitization\_task.   
  
I've attached reference files that show more elements implementation and the reference materials.   
  
Below is a comprehensive assessment and recommendations:

Make a thorough review of the current state of approach, design, architecture, modules, implementations for sanitization\_task.

Once you have completely and thoroughly understood the details. Let me know. I will give you further instructions then.

**Current State Analysis**

The existing SRTPostProcessor has a solid foundation but is limited compared to the sophisticated approaches in your reference code. Key gaps include:

1. **Single-tier hallucination detection** (only exact string matching)
2. **Basic repetition handling** (misses complex patterns)
3. **No cross-subtitle analysis** (each subtitle processed in isolation)
4. **Limited pattern flexibility** (hardcoded Japanese patterns)

**Key Insights from Reference Materials**

Your reference code reveals several advanced techniques:

**From Sample\_code\_1:**

* **Multi-level repetition detection**: Character, word, phrase, and cross-subtitle
* **Sophisticated pattern handling**: Complex grammatical phrases, comma-separated repetitions
* **Cross-subtitle merging**: Detects and combines repetitive consecutive subtitles
* **Fuzzy matching**: Similarity-based hallucination detection
* **NLP integration**: Optional spaCy support

**From Hallucination Data:**

* **Dual approach needed**:
  + Exact string matching (WhisperJAV\_hallucination\_filter\_sorted\_v08.json)
  + Regex patterns with categories (WhisperJAV\_hallucination\_regexp\_v09.json)
* **Language-aware processing**: Different hallucination lists per language
* **Confidence scoring**: Patterns have confidence levels and categories

**Recommended Design**

**1. Rename and Restructure**

I propose renaming SRTPostProcessor to **SubtitleSanitizer** as the main orchestrator, with specialized sub-processors:

# Main orchestrator

class SubtitleSanitizer:

"""Comprehensive subtitle sanitization system"""

def \_\_init\_\_(self):

self.hallucination\_remover = HallucinationRemover()

self.repetition\_cleaner = RepetitionCleaner()

self.cross\_subtitle\_processor = CrossSubtitleProcessor()

self.timing\_adjuster = TimingAdjuster()

# Specialized processors

class HallucinationRemover:

"""Handles exact, regex, and fuzzy hallucination detection"""

class RepetitionCleaner:

"""Multi-level repetition detection and cleaning"""

class CrossSubtitleProcessor:

"""Inter-subtitle analysis and merging"""

class TimingAdjuster:

"""Duration and timing normalization"""

**2. Enhanced Processing Pipeline**

Input SRT

↓

Load Patterns (Exact List + Regex Patterns)

↓

Individual Subtitle Processing

├── Exact hallucination matching

├── Regex pattern matching

├── Fuzzy hallucination detection

└── Advanced repetition cleaning

↓

Cross-Subtitle Analysis

├── Merge consecutive repetitions

├── Detect high-density repetitions

└── Adjust timings

↓

Final Adjustments

├── Duration normalization

├── Renumbering

└── Validation

↓

Output (Sanitized SRT + Reports)

**3. Key Features to Implement**

**A. Dual Hallucination System:**

* Exact match from filter list (fast, 100% confidence)
* Regex patterns with categories and confidence scores
* Fuzzy matching for near-matches

**B. Advanced Repetition Handling:**

* Port the sophisticated patterns from Sample\_code\_1:
  + Comma-separated repetitions (あ、あ、あ...)
  + Grammatical phrase repetitions (私の中で、私の中で...)
  + High-density detection (when one word dominates)
  + Preserve legitimate repetitions (どきどき, わくわく)

**C. Cross-Subtitle Processing:**

* Detect repetitive consecutive subtitles
* Merge based on similarity threshold
* Adjust timing based on reading speed (7 chars/second for Japanese)

**D. Configuration System:**

@dataclass

class SanitizationConfig:

# Hallucination settings

hallucination\_exact\_list\_url: str

hallucination\_regex\_patterns\_url: str

fuzzy\_match\_threshold: float = 0.9

# Repetition settings

repetition\_threshold: int = 2

preserve\_legitimate\_repetitions: bool = True

# Cross-subtitle settings

merge\_similarity\_threshold: float = 0.9

max\_gap\_ms: int = 600

chars\_per\_second: float = 7.0

# Processing levels

aggressiveness: float = 1.0 # 0.5-2.0

# Integration with WhisperJAV sensitivity

sensitivity\_mode: str = "balanced" # conservative/balanced/aggressive

**4. Integration with WhisperJAV**

The sanitizer should integrate seamlessly with your existing sensitivity system:

# In main pipeline

if config.enable\_post\_sanitization:

sanitizer\_config = SanitizationConfig(

sensitivity\_mode=config.sensitivity, # Reuse WhisperJAV sensitivity

# Other params...

)

sanitizer = SubtitleSanitizer(sanitizer\_config)

sanitized\_srt\_path = sanitizer.process(output\_srt\_path)

**5. Architecture Benefits**

This design offers:

* **Modularity**: Each processor can be tested/updated independently
* **Extensibility**: Easy to add new sanitization strategies
* **Compatibility**: Works with WhisperJAV's parameter system
* **Performance**: Can process large files efficiently
* **Transparency**: Detailed reporting of what was changed

**Next Steps**

Would you like me to:

1. Implement the complete SubtitleSanitizer system with all advanced features?
2. Start with a specific component (e.g., the enhanced HallucinationRemover)?
3. Create a detailed integration plan with your existing WhisperJAV pipeline?
4. Design the configuration schema that maps sensitivity levels to sanitization aggressiveness?

The recommended approach would give you a robust, production-ready sanitization system that matches the sophistication of your reference implementations while maintaining clean integration with WhisperJAV.

Excellent suggestion! This transparency feature is crucial for quality control and debugging. I notice your reference code (Sample\_code\_1) already implements a similar approach with the "discarded.srt" file. Let me propose an enhanced version:

**Enhanced File Output Strategy**

**1. File Naming Convention**

Following WhisperJAV's pattern style, I propose:

# Input: video\_name.ja.srt

# Outputs:

video\_name.ja.srt # Sanitized (replaces original)

video\_name.ja.pre-sanitization.srt # Original backup

video\_name.ja.artifacts.srt # Removed content with metadata

Or if preserving original and creating new sanitized:

video\_name.ja.srt # Original (unchanged)

video\_name.ja.sanitized.srt # Sanitized version

video\_name.ja.artifacts.srt # Removed content

**2. Configuration Options**

@dataclass

class SanitizationConfig:

# ... existing fields ...

# Output control

save\_original: bool = True

save\_artifacts: bool = True

preserve\_original\_file: bool = False # If True, creates new file for sanitized

# Naming patterns

sanitized\_suffix: str = "sanitized"

original\_backup\_suffix: str = "pre-sanitization"

artifacts\_suffix: str = "artifacts"

# Artifact report detail level

artifact\_detail\_level: str = "full" # "full", "summary", "minimal"

**3. Artifacts SRT Format**

The artifacts file should be informative, showing what was removed and why:

1

00:00:10,000 --> 00:00:15,000

[REMOVED - Hallucination Match]

Original: "ご視聴ありがとうございました"

Reason: Exact match in hallucination list (category: closing\_phrase)

Confidence: 1.0

2

00:00:45,500 --> 00:00:48,000

[REMOVED - Excessive Repetition]

Original: "あああああああああああ"

Cleaned to: "ああ"

Reason: Repeated vocalization (12 repetitions, threshold: 2)

Pattern: repeated\_vowel\_sounds

3

00:01:23,000 --> 00:01:28,000

[MERGED WITH NEXT 3 SUBTITLES]

Original lines:

- "私の中で、"

- "私の中で、"

- "私の中で、"

- "私の中で、"

Merged to: "私の中で、私の中で、"

Reason: Cross-subtitle repetition (similarity: 1.0)

**4. Implementation Structure**

class SubtitleSanitizer:

def \_\_init\_\_(self, config: SanitizationConfig):

self.config = config

self.removal\_log: List[ArtifactEntry] = []

def process(self, input\_srt\_path: Path) -> SanitizationResult:

"""

Process SRT file and return paths to all generated files

"""

# Setup paths

paths = self.\_setup\_output\_paths(input\_srt\_path)

# Save original if requested

if self.config.save\_original and not self.config.preserve\_original\_file:

shutil.copy2(input\_srt\_path, paths.original\_backup)

# Process subtitles

subtitles = self.\_load\_srt(input\_srt\_path)

sanitized\_subtitles = self.\_sanitize\_subtitles(subtitles)

# Save outputs

output\_path = paths.original if not self.config.preserve\_original\_file else paths.sanitized

self.\_save\_srt(sanitized\_subtitles, output\_path)

if self.config.save\_artifacts:

self.\_save\_artifacts\_srt(self.removal\_log, paths.artifacts)

# Generate summary report

report = self.\_generate\_report()

return SanitizationResult(

sanitized\_path=output\_path,

original\_backup\_path=paths.original\_backup if self.config.save\_original else None,

artifacts\_path=paths.artifacts if self.config.save\_artifacts else None,

report=report

)

@dataclass

class ArtifactEntry:

"""Represents a removed or modified subtitle entry"""

index: int

start\_time: str

end\_time: str

original\_text: str

modified\_text: Optional[str] # None if completely removed

reason: str

category: str

confidence: float

pattern: Optional[str]

additional\_info: Dict[str, Any]

**5. Integration with WhisperJAV Pipeline**

# In WhisperJAV main processing

if self.config.enable\_post\_sanitization:

# Configure based on sensitivity

sanitizer\_config = SanitizationConfig(

sensitivity\_mode=self.config.sensitivity,

save\_original=self.config.save\_sanitization\_original,

save\_artifacts=self.config.save\_sanitization\_artifacts,

preserve\_original\_file=self.config.preserve\_original\_srt,

# Detail level based on debug mode

artifact\_detail\_level="full" if self.config.debug else "summary"

)

sanitizer = SubtitleSanitizer(sanitizer\_config)

result = sanitizer.process(output\_srt\_path)

# Log results

logger.info(f"Sanitization complete: {result.report.summary}")

if result.artifacts\_path:

logger.info(f"Removed artifacts saved to: {result.artifacts\_path}")

**6. Additional Features**

**A. Summary Statistics in Artifacts File:**

1

00:00:00,000 --> 00:00:05,000

[SANITIZATION SUMMARY]

Total Subtitles Processed: 350

Subtitles Modified: 45

Subtitles Removed: 12

Hallucinations Found: 23

Repetitions Cleaned: 18

Cross-subtitle Merges: 4

Processing Time: 0.45s

2

00:00:05,000 --> 00:00:10,000

[CONFIGURATION USED]

Sensitivity: balanced

Repetition Threshold: 2

Fuzzy Match Threshold: 0.9

Aggressiveness: 1.0

**B. Optional JSON Report:**

if self.config.save\_json\_report:

report\_path = input\_path.with\_suffix('.sanitization-report.json')

save\_json({

'summary': report.to\_dict(),

'detailed\_changes': [artifact.to\_dict() for artifact in self.removal\_log],

'configuration': asdict(self.config)

}, report\_path)

**7. Benefits**

This approach provides:

* **Full transparency**: Users can see exactly what was changed
* **Debugging capability**: Easy to identify false positives
* **Rollback option**: Original is preserved
* **Learning tool**: Artifacts file helps understand what patterns are being caught
* **Configurable verbosity**: From minimal to detailed reporting

In addition it would be good idea that the backup original srt and the artefacts srt are saved in a separate subfolder under the output folder, called raw\_subs.