Batch Prediction Framework:
Long Context and Context Caching for Video Analysis

Sean Brar

hello@seanbrar.com | seanbrar.com | github.com/seanbrar

Google Summer of Code 2025 - Google DeepMind Project Proposal

1 Introduction and Background

1.1 Executive Summary

This proposal outlines an efficient and scalable framework for analyzing educational video content using the Gemini API. The approach combines advanced techniques including optimized batch prediction, intelligent context caching, and conversational memory to reduce API usage significantly while improving coherence in multimodal interactions. Inspired by my previous development of ContextRAG, a retrieval-augmented generation system employing adaptive processing strategies, this architecture specifically addresses common limitations encountered when managing extensive and complex video inputs. By integrating these strategies into a cohesive system, the proposal provides both a practical tool for video analysis and a comprehensive demonstration of Gemini API best practices for developers.

1.2 Background and Relevant Experience

As the developer of ContextRAG, a scalable retrieval-augmented generation system, I implemented context-aware processing strategies to optimize semantic retrieval, efficient handling of document collections, and adaptive processing based on content complexity. ContextRAG demonstrated clear advantages in computational efficiency and retrieval accuracy by intelligently adapting approaches based on document length and structure. These experiences directly inform my strategies for effective long-context and multimodal processing in this proposal.

Additional relevant qualifications include: Caltech AI/ML Certificate (2025), experience with Google APIs, strong Python programming skills, and systems engineering background with expertise in scaling solutions.

1.3 Academic and Time Commitments

During the GSoC period (June 16 - August 22, 2025), I will be taking two summer courses at Glendale Community College (Computer Architecture and Assembly Language, Precalculus II). I can dedicate 20+ hours weekly to GSoC with slightly reduced availability around exams. I've specifically chosen this 175-hour project to ensure complete delivery despite academic commitments.

2 Problem Understanding

2.1 Project Context and Goals

This project addresses the challenge of efficiently extracting information from educational videos through natural language queries. When users ask multiple questions about video content, traditional approaches lead to inefficiency (redundant API calls), context fragmentation (long transcripts exceeding model limits), and lack of conversational coherence (follow-up questions without previous context).

For example, a student might ask "What are the three main factors affecting climate change mentioned at 15:23?" followed by "How do these factors interact?" and "Which mitigation strategy was recommended for the second factor?" The goal is to create a solution that: (1) optimizes API usage through batch processing and context caching, targeting a 4× reduction in calls, (2) effectively handles complete transcripts using Gemini's context capabilities, (3) maintains conversation coherence across related questions, and (4) provides clear responses with precise timestamp references.

2.2 Technical Challenges Analysis

The system must address four interconnected technical challenges:

1. **Batch Prediction Optimization**
   - Organizing questions with varying complexity and dependencies into optimal batches
   - Balancing throughput gains against potential quality degradation
   - Implementing efficient asynchronous processing while respecting API rate limits
2. **Long Context Management**
   - Processing transcripts that approach or exceed even Gemini's extended context limits
   - Preserving semantic relationships when chunking is necessary
   - Optimizing token usage while maintaining content integrity
3. **Context Caching Implementation**
   - Developing appropriate caching strategies with optimal TTL parameters
   - Ensuring cache consistency and handling cache misses
   - Managing partial caching for extremely large transcripts
4. **Conversation Coherence**
   - Resolving references to previous answers in follow-up questions
   - Selectively including relevant conversation history without exceeding context limits
   - Maintaining terminology consistency across related responses

2.3 State of the Art and Related Work

Recent advances in LLM optimization provide a foundation for this project:

2.3.1 Batch Processing Techniques

Research by Cheng et al. (EMNLP 2023) demonstrates that batch prompting can reduce inference token counts and latency nearly inverse-linearly with batch size—achieving approximately 5× cost reduction with 6-query batches while maintaining accuracy. These findings directly inform the system's batching strategy.

2.3.2 Long Context Management Approaches

Gemini's context window (up to 1-2 million tokens) enables whole-transcript analysis for most educational videos. According to Google's documentation, Gemini 1.5 achieved >99.8% recall on "needle-in-a-haystack" tests with 1M-token inputs, demonstrating reliable information retrieval across extended contexts. For content exceeding these limits, Retrieval-Augmented Generation (RAG) provides an efficient fallback by extracting relevant transcript segments and linking answers to precise video timestamps.

2.3.3 Context Caching Innovations

Google's Gemini API introduced explicit context caching designed specifically for "repetitive analysis of lengthy video files." This feature allows caching of large blocks of text under persistent IDs for configurable periods (default: 1 hour), significantly reducing both token costs and latency for repeated queries. The proposed system leverages this capability as a cornerstone of applicable efficiency optimizations, while accounting for Gemini's minimum cache size requirement (32K tokens).

2.3.4 Conversation Memory Management

To combat "Context Degradation Syndrome," where models gradually "lose the plot" in extended conversations (Howard, 2024), the framework employs a hybrid approach using summarization for older exchanges while maintaining full context for recent and highly relevant interactions. This strategy, supported by research from VerticalServe and implemented in frameworks like LangChain, provides the optimal balance between conversation coherence and token efficiency.

2.4 Synthesis

While these techniques have been explored separately, their integration into a comprehensive system for video content analysis represents a novel contribution that addresses unique challenges not solved by any single technique alone. By leveraging Gemini's multimodal understanding and context-handling capabilities, this architecture will deliver a reference framework that demonstrates best practices for efficient, user-friendly, and cost-effective AI-powered video analysis. The project includes benchmarking components to measure performance improvements across various metrics: API call reduction, latency improvement, token efficiency, and answer quality compared to non-optimized approaches.

3 Technical Approach

3.1 System Architecture

The system follows a modular architecture designed to leverage Gemini API's capabilities while addressing the specific requirements for batch prediction, long context handling, and efficient interconnected question processing:

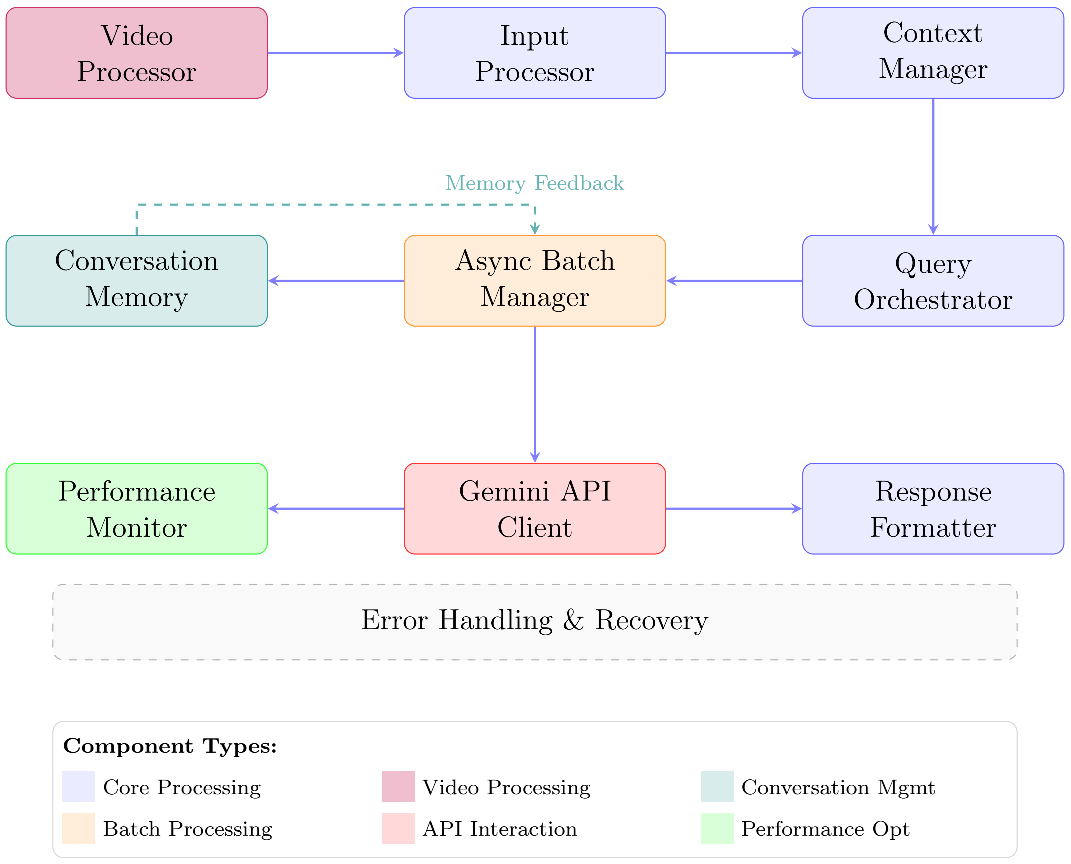

Figure 1: System Architecture for Batch Prediction with Long Context

The system is composed of nine specialized components:

1. **Video Processor**: Handles video input, transcript generation, and timestamps extraction
2. **Input Processor**: Normalizes transcript text and analyzes questions for dependencies
3. **Context Manager**: Implements context caching and selects optimal strategies based on transcript length.
4. **Query Orchestrator**: Groups questions for efficient batching while respecting dependencies
5. **Async Batch Manager**: Optimizes asynchronous processing for independent question batches.
6. **Conversation Memory**: Maintains state for interconnected questions to ensure coherent follow-ups.
7. **Gemini API Client**: Provides efficient interface to Gemini's capabilities with authenti-

cation and caching.

8. **Response Formatter**: Creates structured outputs with video timestamp references
9. **Performance Monitor**: Tracks token usage and optimizations to measure efficiency gains

Cross-Cutting Concern: The Error Handling & Recovery system spans all components, providing input validation, error classification, exponential backoff for transient failures, and graceful degradation paths when primary strategies encounter limitations.

3.2 Model Selection Rationale

This system strategically uses Gemini 1.5 Pro as the primary model despite the availability of newer Gemini 2.0 and 2.5 models (released in Q1 2025). This decision is based on several technical considerations:

1. **Context Caching Support**: Gemini 1.5 Pro provides explicit context caching capabilities, which are essential for this project's efficiency goals. As of April 2025, the newer models do not support this feature, which is central to achieving the targeted 4-5× reduction in API calls.
2. **Extended Context Window**: Gemini 1.5 Pro offers an input token limit of 2,097,152 tokens, approximately twice that of other models. According to Google's documentation, this allows for processing videos up to two hours in length without chunking, which significantly simplifies transcript handling for most educational content.
3. **Cost-Performance Balance**: While Gemini 1.5 Pro has higher per-token costs than Gemini 1.5 Flash, the context caching feature ultimately reduces overall token consumption by avoiding redundant processing of the same transcript for multiple questions.

The architecture is designed with modularity to facilitate migration to newer models when context caching becomes available for them, ensuring the system remains adaptable to evolving API capabilities.

3.3 Key Implementation Highlights and Optimization Benefits

The system focuses on five core technical components that enable efficient video content analysis with Gemini's API capabilities, delivering significant optimization benefits.

3.3.1 Context Caching Implementation

For efficient repeated access, the system implements Gemini's context caching API to store transcript content:

```python
def cache_transcript_content(transcript, ttl="3600s"):
    """Cache transcript segment in Gemini's context cache"""
    display_name = f"transcript_{hashlib.md5(transcript.encode()).hexdigest()[:16]}"

    cache = client.caches.create(
        model="models/gemini-1.5-pro-001",
        config=types.CreateCachedContentConfig(
            display_name=display_name,
            contents=transcript,
            ttl=ttl,
        )
    )
    return cache
```

The caching system implements whole-transcript caching for content under 125K tokens and segmented caching with variable TTL settings for longer transcripts, optimizing both performance and cost.

3.3.2 Token Tracking and Optimization

The architecture uses Gemini's direct token counting capabilities to make informed decisions about context handling:

```python
def select_context_strategy(transcript, client):
    """Select appropriate context strategy based on transcript length"""
    token_count_response = client.models.count_tokens(
        model="models/gemini-1.5-pro-001",
        contents=transcript
    )
    token_count = token_count_response.total_tokens

    if token_count < 8000:
        return {"strategy": "standard_context", "model": "models/gemini-1.5-flash-001"}
    elif token_count < 125000:
        return {"strategy": "long_context", "model": "models/gemini-1.5-pro-001"}
    else:
        # For extremely large transcripts (>125K tokens), use chunking with RAG
        return {"strategy": "chunked_retrieval", "model": "models/gemini-1.5-pro-001"}
```

This approach provides precise usage metrics for cost optimization and enables intelligent model selection based on content length.

3.3.3 Batch Processing Implementation

The system optimizes question processing through intelligent batching, combining multiple questions into a single API call:

```python
def process_question_batch(client, batch, cache):
    """Process a batch of independent questions efficiently"""
    formatted_prompt = "Answer each of the following questions about the video:\n\n"

    for i, question in enumerate(batch, 1):
        formatted_prompt += f"Question {i}: {question['text']}\n"

    # Make a single API call with the batched questions
    response = client.models.generate_content(
        model="models/gemini-1.5-pro-001",
        contents=formatted_prompt,
        config=types.GenerateContentConfig(cached_content=cache.name)
    )
    return response
```

This approach significantly reduces API call overhead while maintaining answer quality, with batch sizes dynamically selected based on token count constraints.

3.3.4 Video Processing Integration

Direct video processing leverages Gemini's multimodal capabilities through the File API, with accurate tracking of video token usage (263 tokens per second):

```python
def process_video_file(video_path):
    """Upload and process a video file using Gemini's File API"""
    video_path = pathlib.Path(video_path) if isinstance(video_path,
   str) else video_path

    # Upload the video using the Files API
    video_file = client.files.upload(file=video_path)

    # Wait for the file to finish processing
    while video_file.state.name == 'PROCESSING':
        time.sleep(2)
        video_file = client.files.get(name=video_file.name)

    return video_file
```

3.3.5 Structured Output Implementation

The system generates consistent, machine-readable JSON responses using Gemini's structured output capabilities:

```python
from pydantic import BaseModel
from typing import List, Optional, Dict

class VideoAnswer(BaseModel):
    """Model for structured video answers"""
    question: str
    answer: str
    question_id: Optional[int]
    timestamps: List[Dict[str, str]]
    metadata: Dict[str, any]

def generate_structured_response(result_data):
    """Generate a structured JSON response"""
    response = client.models.generate_content(
        model="models/gemini-1.5-pro-001",
        contents=f"Convert the following result data into JSON: {result_data}",
        config={
            'response_mime_type': 'application/json',
            'response_schema': List[VideoAnswer],
        },
    )
    return response.text
```

3.4 End-to-End Processing Flow

The complete system integrates these components into a cohesive pipeline that processes video uploads, generates and caches transcripts, analyzes question dependencies, processes questions in optimized batches, and returns structured responses with timestamp references. This approach delivers a 4–5× reduction in API calls while maintaining conversation coherence across multiple related questions.

3.5 Key Benefits

This architecture achieves the following improvements:

- **API Usage Optimization:** Intelligent batch processing and context caching enable a targeted 4–5× reduction in API calls and associated costs.
- **Long Context Handling:** Leveraging Gemini's million-token context capabilities for handling even multi-hour educational videos.
- **Accurate Token Tracking:** Direct token counting instead of estimation, providing precise usage metrics for cost optimization.
- **Video Token Optimization:** Specialized handling for video content (which consumes 263 tokens/second) with exact duration detection.
- **Enhanced User Experience:** Structured responses with direct links to relevant video timestamps and consistent formatting.

This implementation serves as both a practical tool for analyzing educational video content and a reference architecture demonstrating Gemini API best practices for multimodal content analysis.

4 Implementation Plan

4.1 Project Timeline

The project will be implemented over the 13-week GSoC period, with a phased approach to ensure consistent progress and regular deliverables.

4.1.1 Community Bonding Period (May 2025)

- Set up development environment and GitHub repository

- Study Gemini API documentation and test basic functionality
- Establish communication channels and evaluation metrics with mentor
- Create preliminary design documents and project structure

4.1.2 Phase 1: Foundation (June 2 - June 23)

- **Week 1: Core Architecture Implementation**
  Implement basic system architecture, input processing module, and testing framework
  Deliverable: Repository with basic architecture and module structure
- **Week 2: API Integration Layer**
  Develop Gemini API client with authentication, retry logic, and error handling
  Deliverable: Working API client with test suite
- **Week 3: Batch Optimization & Processing**
  Develop batch optimization algorithms with dependency analysis and processing utilities
  Deliverable: Basic batch processing implementation

4.1.3 Phase 2: Core Features (June 24 - July 21)

- **Week 4: Context Caching Implementation**
  Implement context caching module with cache creation, management, and optimization utilities.
  Deliverable: Working context caching system with tests
- **Week 5: Context Classification and Chunking**
  Build transcript classification system with chunking strategies for different transcript lengths.
  Deliverable: Working transcript processing system
- **Week 6: Conversation Memory & Question Dependencies**
  Create conversation memory manager with dependency tracking and relationship analysis
  Deliverable: Working conversation memory system
- **Week 7: Response Formatting & Timestamp Handling**
  Build response enhancement with timestamp extraction, linking, and structured output generation.
  Deliverable: Complete response formatting system

4.1.4 Phase 3: Optimization & Testing (July 22 - August 18)

- **Week 8: Performance Optimization**
  Implement asynchronous batch processing and optimize token usage and API calls
  Deliverable: Optimized processing pipeline
- **Week 9: Error Handling & Resilience**
  Enhance error recovery mechanisms and implement graceful degradation
  Deliverable: Robust error handling framework
- **Week 10: Comprehensive Testing**
  Create extensive test suite with varied content and benchmark comparisons

Deliverable: Comprehensive test results and metrics

- **Week 11: Documentation & Examples**
  Create comprehensive API documentation, detailed setup instructions (including obtaining and configuring API keys), practical usage examples with representative questions and expected outputs, and configuration guides.
  Deliverable: Complete, detailed documentation including API configuration and fully commented example code.

4.1.5 Final Phase: Polishing (August 19 - September 1)

- **Week 12: Code Cleanup & Final Optimizations**
  Perform code review, refactoring, and implement final optimizations
  Deliverable: Clean, optimized codebase
- **Week 13: Final Documentation & Delivery**
  Update documentation, create demo materials, prepare contribution guidelines
  Deliverable: Complete project with documentation

5 Community Benefits and Conclusion

5.1 Benefits to DeepMind and Developer Community

This solution will provide significant value to both DeepMind and the broader developer ecosystem:

- **Showcase for Gemini Capabilities**: Demonstrates Gemini's advanced features in a real-world application that drives API adoption.
- **Reference Implementation**: Provides a complete example of best practices for batch prediction, context caching, and long-context processing.
- **Educational Resource**: Well-documented code with performance insights helps developers make informed architectural decisions.
- **Practical Solutions**: Reusable patterns and utilities for common challenges in video content analysis.

5.2 Conclusion

The proposed system provides a comprehensive and novel solution for efficiently analyzing video content with LLMs by uniquely integrating optimized batch prediction, adaptive context caching, and conversational memory management into a unified framework. This integration effectively addresses critical limitations that current standalone approaches fail to solve, enabling significant improvements in API efficiency, conversational coherence, and long-context processing. By making strategic model selections based on feature requirements rather than just recency, and implementing intelligent rate limit management for real-world deployment considerations, the system achieves practical optimization across multiple dimensions. Drawing directly on my prior experience developing ContextRAG, this modular and extensible architecture demonstrates best practices while fully showcasing Gemini's advanced capabilities for multimodal content analysis.