Google DeepMind

Batch Prediction with Long Context and Context Caching Code Sample
Google Summer of Code 2025 Proposal

Organization: Google DeepMind
Name: Vansh Kumar Singh
Email: vsvsasas@gmail.com
Location: Jaipur, Rajasthan, India
Preferred Communication Method: Email, Video Conferencing, or Call
Project Title: Batch Prediction with Long Context and Context Caching Code Sample

🌟 About Me

I am a 3rd-year B.Tech (Hons) Computer Science student specializing in IoT & Intelligent Systems at Manipal University Jaipur (2022–2026). My passion lies in building real-world, production-grade AI systems that are intelligent, interactive, and impactful.

Over the past few years, I've led and contributed to several high-traction open-source projects that combine GenAI with edge computing and agentic AI, including:

- **Gemini API M5Cardputer**: A portable AI assistant using Gemini APIs on an M5Stack Cardputer — **2,000+ downloads**, **40+ GitHub stars**, published on **M5Burner**.
- **Pi5Neo**: A high-performance NeoPixel SPI library for Raspberry Pi 5 — **2,000+ downloads**, **30+ GitHub stars**, available on **PyPi**.
- **Ascendant AI** and **MedBuddy AI**: Agentic AI prototypes built with Gemini and LangChain for real-time decision-making and contextual memory.

With **8 hackathon** wins, **6 patents filed**, and experience across organizations like **NTPC Limited**, **Bluestock Fintech**, and **Solar Secure**, I bring both engineering depth and product thinking to the table. My recent work includes retrieval-augmented generation (RAG) systems, long-context Gemini pipelines, and local LLM assistants like **Manipal-LLM**.

🧠 Technical Strengths

- **Languages & Tools**: Python, Embedded C, Node.js, Git, CI/CD, GCP, AWS
- **AI/ML & GenAI**: Gemini, LangChain, Hugging Face, RAG pipelines, LLMOps
- **Web & UI**: Streamlit, HTML/CSS/JS, API Integration, Web Scraping
- **Edge/Embedded**: Raspberry Pi, Arduino, M5Stack, IoT, RTOS, Edge LLMs

Why DeepMind & This Project?

DeepMind’s pursuit of safe, responsible, and capable AI resonates deeply with my personal mission. I’ve long believed that the **future of LLMs lies beyond the cloud**—in offline-first, edge-ready, and agent-driven deployments.

In fact, I’ve already been building in this direction:

- **Ascendant AI** and **MedBuddy AI** explore agentic AI for real-world workflows.
- **Manipal-LLM** brings on-device NLP to students and researchers.
- My custom **RAG + retrieval strategy experimentation system** explores Self-RAG, Adaptive Retrieval, and BERTScore/METEOR evaluation—all compatible with Gemini and LangChain.

These experiences have prepared me to contribute to projects around:

- Gemini/Gemma model optimization
- Fine-tuning interfaces for OSS users
- Retrieval-augmented generation & context caching
- Integrations with LangChain, LlamaIndex, CrewAI, and other open agent tools

🌐 Availability

I’m fully available for **40–50 hours/week throughout the summer**, except for a short academic break (End Term Evaluations) in early May (2–3 hrs/day). I have **no conflicting internships** and will prioritize GSOC fully. I bring experience in team collaboration, agile standups, milestone tracking, and timely communication.

🚀 What I Hope to Achieve

For me, GSoC with DeepMind isn’t just a coding project—it’s a **milestone in shaping responsible and accessible AI**. I aim to:

- Contribute to open-source GenAI/LLM ecosystems
- Learn from top researchers and engineers
- Deliver tools that push Gemini’s reach—on the edge, for everyone

🔗 Important Links

- **GitHub:** https://github.com/vanshksingh

  A collection of repositories showcasing projects in AI, hardware integration, and software development. It includes tools for medical assistance, gaming platforms, and advanced APIs.

- **LinkedIn:** https://www.linkedin.com/in/vansh-kumar-singh-711924204/

Professional profile highlighting achievements, skills, and contributions in AI, software development, and engineering.

- **LeetCode:** https://leetcode.com/u/vanshksingh/

  Profile showcasing problem-solving expertise with 245 problems solved in Python. Skills include dynamic programming, hash tables, and divide-and-conquer techniques.

- **PyPi – Pi5Neo:** https://pypi.org/project/Pi5Neo/

  A Python library designed for controlling NeoPixel LED strips on Raspberry Pi 5. It simplifies creating vibrant lighting effects using SPI interface.

- **Gemini API M5Cardputer:** https://github.com/vanshksingh/M5Cardputer-Chat-with-Gemini-API

  An application enabling user interaction with the Gemini API via M5Cardputer. Features include Wi-Fi connectivity, user input handling, and response display.

- **Manipal-LLM:** https://github.com/vanshksingh/Manipal_LLM

  Repository focused on building a large language model tailored for specific applications.

- **Ascendant AI:** https://github.com/vanshksingh/Ascendant_Ai

  Part of a gamified AI platform integrating tokenization and NFTs to create interactive gaming experiences powered by AI.

- **MedBuddy AI:** https://github.com/vanshksingh/MedBuddy-OnDeviceAI

  An AI-powered medical assistant that ensures privacy by performing tasks on-device. It supports drug interaction checks, image processing, and natural language conversations.

- **Multi_RAG:** https://github.com/vanshksingh/RAG_Type

  A repository implementing Various Retrieval-Augmented Generation (RAG) techniques for enhancing AI-driven retrieval processes.

🔧 Specifications and Introduction

The goal of this project is to create a high-quality, production-grade code sample that demonstrates **batch prediction over long-context inputs using Google Gemini APIs**. The primary use case will be **multi-turn question answering on long-form video content**, such as lectures or documentaries.

The solution will showcase how to leverage **context caching**, **long-context optimization**, and **interconnected query handling**, making it an excellent resource for developers building AI-first educational or media summarization tools.

🎯 Core Project Goals

These are the deliverables I aim to complete within the GSoC timeline:

1. **Batch Prediction Logic**
   - Build a modular pipeline to submit batches of related queries to Gemini while minimizing token and API overhead.
   - Handle async processing using Python's `asyncio` or concurrent futures.
2. **Long Context Handling**
   - Support full or partial transcripts of videos/documents as input context.
   - Implement chunking/fallback for transcripts that exceed Gemini's context limit, preserving semantic boundaries.
3. **Context Caching System**
   - Cache processed context (tokenized or full) to reuse across batch runs, reducing API calls and speeding up response generation.
   - Use in-memory and optional file-based storage.
4. **Interconnected Question Support**
   - Implement conversation thread tracking to let later questions use previous answers.
   - Enable simple dialogue memory within the batch or across sessions.
5. **Output Formatter**
   - Display answers cleanly in JSON/Markdown with optional links to video timestamps.
   - Include indicators for reference sections or reused context.
6. **Documentation and Examples**
   - Well-documented Jupyter notebooks and Python scripts.
   - Setup instructions, sample transcript + questions, and walkthroughs.
7. **Robust Error Handling & Testing**
   - Handle Gemini API errors, empty responses, malformed input.
   - Add test cases for different batch sizes, contexts, and caching scenarios.

🌟 Stretch Goals (Time-Permitting Enhancements)

If time permits and core milestones are complete, I propose implementing the following enhancements:

1. **Live Gemini Streaming for Better UX**
    - Enable partial streaming of answers as they are generated, for real-time UX in interactive shells or UIs.
2. **Semantic-Aware Chunking**
    - Use sentence transformers or Gemini embeddings to split transcripts semantically before input, ensuring high-quality context management.
3. **Basic Frontend Demo (Streamlit or CLI)**
    - Wrap the functionality in a Streamlit app or CLI demo that accepts transcript + batch questions and returns structured answers.
4. **Thread Persistence**
    - Save and reload ongoing Q&A sessions by assigning a thread ID.
    - Reuse stored conversation state for continuous work.
5. **Inference Efficiency Tracker**
    - Print estimated token cost, runtime per batch, and average response latency.

⚙️ Tools & Tech Stack Summary

Tool/Library	Purpose
Google Gemini API	Primary LLM for all prediction tasks and streaming responses
Python (asyncio)	Asynchronous batch processing and concurrency
Streamlit (optional)	UI demo for developers to test inference workflows
Sentence Transformers	Semantic chunking of long transcripts
diskcache / pickle	Context caching (in-memory and disk-based)
Pytest	Unit testing of pipeline components and error scenarios
Jupyter Notebooks	Interactive examples, experiments, and documentation
draw.io	Architecture diagrams and workflow visualizations

🏗️ Architecture Table

Module	Description	Tech Stack
Batching Engine	Processes and dispatches multiple user queries together, minimizing API calls while maximizing throughput.	Python, `asyncio` / `concurrent.futures`
Long-Context Handler	Splits long transcripts into semantically coherent chunks and handles Gemini context limits.	Gemini, Sentence Transformers (Stretch)
Context Cache	Caches tokenized or processed context to avoid recomputation in future batches.	`diskcache`, `joblib`, or simple in-memory dict
Q&A Interlinking Module	Tracks references between questions in a batch for conversational flow and state retention.	Custom thread/session logic
Output Formatter	Converts Gemini output into JSON/Markdown format with metadata, timestamps, etc.	Markdown, `json`, `re`
Streaming Interface (Stretch)	Enables partial Gemini outputs to be streamed in real-time to a CLI or UI.	Gemini Streaming API, WebSockets (optional)
Frontend Demo (Stretch)	A Streamlit or CLI app for user interaction with transcript input, batch Q&A, and results preview.	Streamlit, CLI
Testing Suite	Unit and integration tests for async logic, batching, and caching scenarios.	`pytest`, `unittest`
Docs & Notebooks	Setup guides, walkthrough notebooks, and example use cases.	Jupyter, Markdown, GitHub Pages

⚠️ Risk Mitigation Table

Risk	Mitigation
Gemini API rate limits or token overflows	Implement batch size control, fallback retry logic, and error parsing. Provide usage cost estimates in docs.
Transcript too long for Gemini context	Use chunking with semantic fallback. Provide warnings for dropped or truncated context.
High token cost or latency	Integrate context caching and async batch handling. Include inference efficiency tracker for user transparency.
Batch processing race conditions or failures	Use robust async patterns and unit tests for concurrent task flows. Handle edge cases like empty/malformed input.
User confusion with JSON/Markdown output	Provide schema templates and clean output formatting. Include visualization options in frontend demo.
API changes in Gemini (especially for streaming)	Keep abstraction layer modular for easy updates. Document version dependencies.
Frontend stretch goals slow progress	Prioritize CLI demo first. Move Streamlit to stretch with minimal viable UI.
Limited access to real videos/transcripts	Use public datasets like YouTube captions, OpenLectures, or synthetic transcripts for testing/demo.

🛠️ Implementation Plan

This section outlines the concrete technical roadmap for implementing both the core features and time-permitting enhancements of the project. The approach emphasizes modularity, performance optimization, and maintainability while showcasing best practices for working with Gemini APIs.

🎯 Core Project Goals

1. 🔄 Batch Prediction Logic

Objective: Efficiently handle batch queries with minimal token and API overhead.
Implementation Steps:
• Design a `BatchPredictor` class to encapsulate batch handling logic.
• Use `asyncio` or `concurrent.futures` for concurrent API requests.
• Token-checking logic to batch only as much as Gemini context can support.
• Integrate rate-limit and retry policies for batch safety.

```
async def batch_predict(queries, context):
    tasks = [gemini_call(q, context) for q in queries]
    return await asyncio.gather(*tasks)
```

2. 📏 Long Context Handling

Objective: Enable large transcripts (e.g., 10k+ words) to be processed efficiently by the Gemini model.
Implementation Steps:
• Create a tokenizer-aware chunker that segments transcripts near semantic boundaries (e.g., paragraph ends).
• Build a fallback summarizer for when even chunked input is too large.
• Inject chunk metadata to allow partial recombination during post-processing.

```
chunks = smart_chunk(transcript, max_tokens=2048)
if too_large(chunks): chunks = [summarize(c) for c in chunks]
```

3. 💾 Context Caching System

Objective: Reuse processed context or tokenized input across sessions and batch calls.
Implementation Steps:
• Implement a hybrid cache system using:
  o In-memory (`lru_cache`, `dict`)
  o Persistent cache (JSON-based key-value or SQLite)
• Use transcript hash and query fingerprinting to validate cache entries.
• Include cache warm-up logic for large input sets.

```
@lru_cache(maxsize=256)
def get_context(key): return db.get(key)
```

4. 🔗 Interconnected Question Support

Objective: Allow the batch engine to reference prior answers, enabling a more dialogue-like thread.
Implementation Steps:
• Implement a `ConversationSession` class to manage session-scoped memory.
• Inject relevant past Q&A snippets into current prompt context.
• Use cosine similarity (`sentence-transformers` or Gemini embeddings) to link related questions.

```
if cosine_sim(q1_embed, q2_embed) > 0.85:
    context += previous_answer
```

5. ✨ Output Formatter

Objective: Provide user-friendly and structured outputs for analysis, debugging, or UI integration.
Implementation Steps:

• Create Markdown and JSON renderers.
• Annotate responses with:
  o Original question
  o Source chunk or context reference
  o Optional: Estimated token usage and latency
• Implement timestamp linking for video-based transcripts (e.g., `00:04:23` jump links).

```
output = {
    "q": question,
    "a": answer,
    "chunk_id": source_id,
    "tokens": est_tokens
}
```

6. 📚 Documentation and Examples

Objective: Make the project easy to use, extend, and learn from.
Implementation Steps:
• Add a comprehensive `README.md` with:
  o Setup and dependency installation
  o Gemini API key configuration
  o Sample transcripts and questions
• Provide:
  o A Jupyter notebook walkthrough
  o A CLI script for batch runs
  o Example outputs

```
python run_batch.py --input questions.json --transcript notes.pdf
```

7. 🛡 Robust Error Handling & Testing

Objective: Build confidence in system stability across diverse usage scenarios.
Implementation Steps:
• Handle Gemini API errors (timeouts, rate limits, input validation) with exponential backoff.
• Write test suites for:
  o Batching logic
  o Cache management
  o Output rendering
• Use mocked API calls in CI/CD-friendly test harness.

```
try:
    result = gemini_call(input)
except RateLimitError:
    time.sleep(2); retry(input)
```

🌟 Stretch Goals (Time-Permitting Enhancements)

1. 🔄 Live Gemini Streaming for Real-Time UX

Objective: Provide streamed output for faster visual feedback.
Implementation Steps:
• Use Gemini's streaming API endpoint (if available).
• Wrap response generator in a line-by-line yield stream.
• Display streaming results in CLI or Streamlit via `st.text()` updates or animated markdown blocks.

```
for line in stream_response(prompt):
    st.markdown(line)
```

2. 🧠 Semantic-Aware Chunking

Objective: Improve transcript slicing to reduce loss of meaning.
Implementation Steps:
• Use `sentence-transformers` or Gemini embeddings to vectorize and cluster transcript sentences.
• Segment on topic shifts or discourse markers.
• Label chunks for improved traceability and cache reusability.

```
embeds = embed_sentences(sentences)
clusters = cluster_embeddings(embeds)
```

3. 🖥️ Basic Frontend Demo (Streamlit or CLI)

Objective: Demonstrate project capabilities to non-technical users and mentors.
Implementation Steps:
• Create a basic Streamlit interface:
  o Upload transcript (PDF, TXT, JSON)
  o Add question list (manually or via upload)
  o Display output with progress indicators
• Optionally export results as Markdown, CSV, or JSON.

```
st.file_uploader("Upload transcript")
st.text_area("Enter questions")
st.button("Run Q&A")
```

4. 📂 Thread Persistence

Objective: Allow users to resume previous Q&A sessions.
Implementation Steps:
• Assign UUIDs to sessions and store context + history.

• Save session metadata (timestamp, topic, length) to persistent store.
• Add CLI flag or Streamlit dropdown to restore sessions.

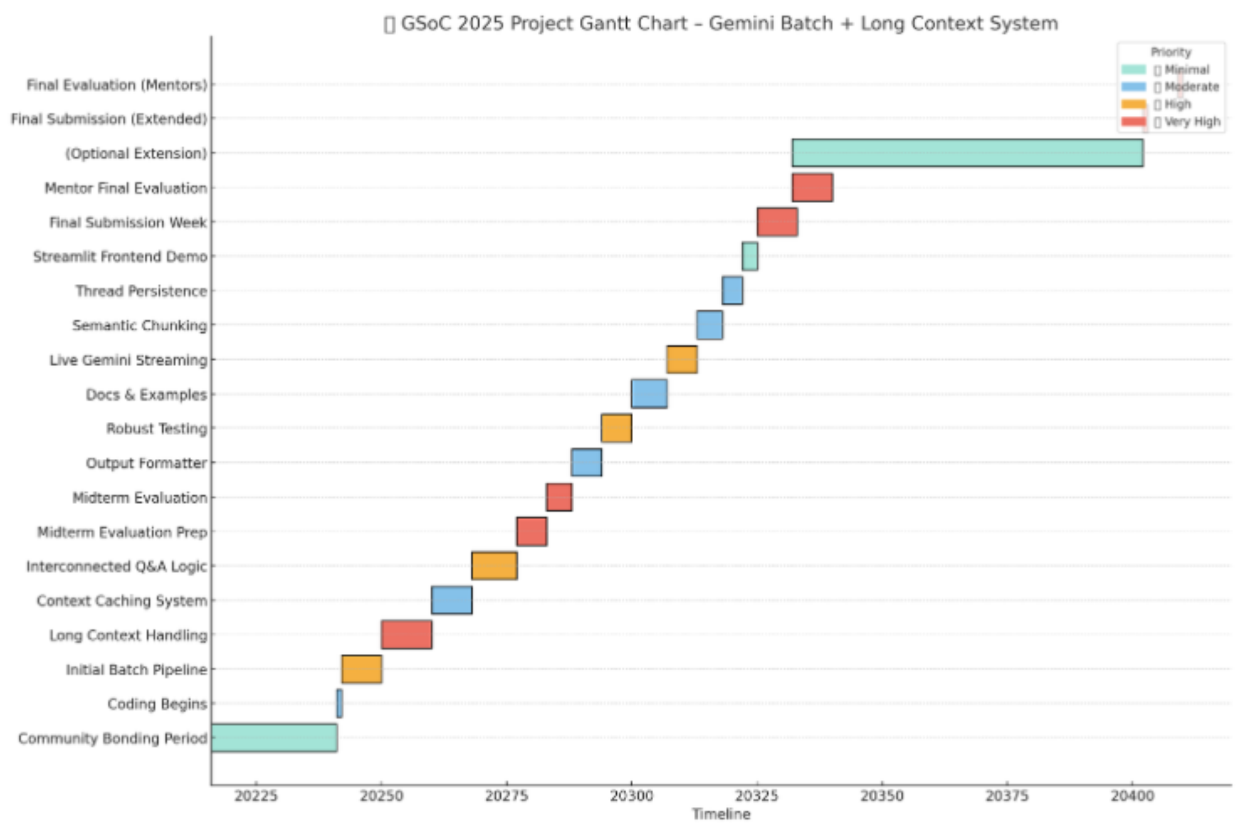
```
session_id = uuid4()
save_session(session_id, context, history)
```

5. 📊 Inference Efficiency Tracker

Objective: Help users optimize cost and performance trade-offs.
Implementation Steps:
• Log and display:
  o Total tokens used
  o Inference time per batch
  o Average latency
• Provide summary reports after each batch run.

```
print(f"Tokens used: {tokens}, Time: {elapsed:.2f}s")
```

📅 Timeline / Project Plan



Timeline: *Gemini Batch+Long Context System*

Time Frame	Start Date	End Date	Task	Priority
Community Bonding Period	May 8	June 1	Understanding Gemini APIs, setting up local dev env, initial doc review	🟢 Minimal
Coding Begins	June 2	June 2	Project Kickoff: Basic script scaffold, environment setup	🔵 Moderate
Initial Batch Pipeline	June 3	June 10	Async batch prediction logic using `asyncio`, error handling base	🟠 High
Long Context Handling	June 11	June 20	Token-aware transcript chunking, Gemini-compatible long-context injection	🔴 Very High
Context Caching System	June 21	June 28	In-memory + disk-based caching of Gemini context payloads	🔵 Moderate
Interconnected Q&A Logic	June 29	July 7	Basic memory threading and previous answer referencing	🟠 High
Midterm Evaluation Prep	July 8	July 13	Prepare deliverables, polish code, complete core milestone summary	🔴 Very High
Midterm Evaluation	July 14	July 18	**Phase 1 Evaluation submission period**	🔴 Very High
Output Formatter	July 19	July 24	Markdown/JSON output structure, link answers to timestamps	🔵 Moderate
Robust Testing	July 25	July 30	Edge cases, Gemini API errors, invalid inputs, full test coverage	🟠 High
Docs & Examples	July 31	August 6	Jupyter notebooks, usage scripts, setup instructions	🔵 Moderate
Live Gemini Streaming	August 7	August 12	Enable real-time answer stream display	🟠 High
Semantic Chunking	August 13	August 17	Split long inputs by meaning using sentence transformers	🔵 Moderate
Thread Persistence	August 18	August 21	Save/load conversations with unique thread IDs	🔵 Moderate
Streamlit Frontend Demo	August 22	August 24	Lightweight UI demo using Streamlit	🟢 Minimal
Final Submission Week	August 25	September 1	Final project zip, complete documentation, contributor eval submission	🔴 Very High
Mentor Final Evaluation	September 1	September 8	Mentor submits evaluation of project	🔴 Very High
(Optional Extension)	September 1	November 9	Further improvements, community feedback incorporation	🟢 Minimal

Final Submission (Extended)	November 10	November 10	**Absolute last date** for submission (if extended)	🔴 Very High
Final Evaluation (Mentors)	November 17	November 17	Mentor finalizes evaluation for extended projects	🔴 Very High

📌 Success Metrics

These clear goals will help evaluate progress and effectiveness by the end of GSoC:

- ✅ Complete core pipeline for batch question answering with Gemini
- ✅ Integrate token-aware long-context chunking and fallback mechanisms
- ✅ Implement in-memory and disk-based context caching
- ✅ Enable basic inter-question memory and dialogue threading
- ✅ Provide 3+ end-to-end example notebooks or scripts
- ✅ Achieve 90%+ test coverage for all major pipeline components
- ✅ Provide at least 1 Streamlit or CLI demo for hands-on interaction
- ✅ Receive feedback from at least 5 early testers or devs

🧠 Developer Persona: Dev Flow Snapshot

Imagine this scenario:

A developer at an edtech startup wants to build a Q&A system that can summarize video lectures using Gemini. With my project as a base, they can:

1. Clone the GitHub repo
2. Drop in a `.txt` file of a lecture transcript
3. Add a list of questions in a YAML or JSON file
4. Run one command to get batch answers with markdown output and timestamps
5. Use the Streamlit demo to view answers or share a prototype internally

This flow ensures even beginner developers can start using Gemini for long-context educational tasks **within minutes**.

🔚 Wrapping Up – Why This Project, Why Me?

I am genuinely excited about the opportunity to contribute to this project under the guidance of the exceptional team at Google DeepMind. If selected, I will dedicate myself fully to ensuring the success of this initiative and making substantial contributions throughout the GSoC period — and beyond.

This project sits at the intersection of everything I'm passionate about: powerful AI systems, clean developer tooling, and open collaboration. I'm deeply inspired by the groundbreaking work within DeepMind's open-source ecosystem and would be thrilled to continue contributing to related projects long after GSoC ends.

I also believe in giving back. As someone who's benefited from open-source communities and mentorship, I'm passionate about helping others get started too—whether through documentation, community support, or even mentoring future GSoC contributors.

That said, even if I'm not selected this year, my commitment to this project and DeepMind's open-source mission remains unchanged. I will continue contributing where possible and return with even more experience and enthusiasm next year.

Looking forward to the opportunity to learn, build, and grow with the community.

Warm regards,
Vansh Kumar Singh