



# Google Summer of Code 2025: Open Climate Fix

## 1) Project Information

Project Title: **Streaming Zarr from Cloud Storage (Ice Chunk + Zarr 3)**

### Project Description:

We use a large amount of Satellite and Numerical Weather Prediction data all saved in Zarr format for training our ML models. We normally have a local copy, rather than using the cloud. We would like to explore using Ice Chunk and Zarr 3. It would be great to create a benchmark when training our PVNet model with data in cloud storage (using modern stack like Ice Chunk + Zarr 3). We could then use this to measure speed and compare it to training using data on disk.

### Expected Outcome:

A quantitative comparison between the speeds when using these new tools and not.

### Other Key Information

- Expected Size: 175hrs
- Skills: ML Knowledge, Familiarity with training ML Models with Pytorch, Python, Data Analysis. Zarr is a bonus
- Difficulty level: Medium
- RelatedReading: <https://github.com/openclimatefix/Satip/issues/222>, <https://github.com/orgs/openclimatefix/discussions/29>

## 2) Contributor Information

- Name: **Dakshbir Singh Kapoor**
- Email: [dakshbirkapoor@gmail.com](mailto:dakshbirkapoor@gmail.com)
- GitHub: [Dakshbir](#)
- LinkedIn: [Dakshbir Singh Kapoor](#)

### Birthplace

- Location: Delhi, India
- TimeZone: IST (GMT+5:30)

### Education

- University: Netaji Subhash University Of Technology (2023-2027)
- Degree: Bachelor of Technology
- Branch Of Study : Computer Science Engineering
- Current year: 2nd (4<sup>th</sup> semester)
- CGPA: 8.67/10

## 3) Potential Mentor(s)

Primary Mentor: [sol@openclimatefix.org](mailto:sol@openclimatefix.org), [peter@openclimatefix.org](mailto:peter@openclimatefix.org)

## 4) Open Source Contributions

Although I have made a number of pull requests but, I would like to highlight some of the best contributions that I opened, in which I also have implemented suggestions from mentors:

### ● Merged PRs- **Open Climate Fix**

- **#409: Dynamic Model Selection**- Introduced model selection in the Solar Forecast API, empowering users with flexible forecasting options.
- **#341: Enhanced Model Versioning**- Added a hub\_branch parameter for better model versioning in Hugging Face uploads.
- **#193: Improved README**- Added the Adjuster Model section and a troubleshooting guide.

### ● Open PRs- **Open Climate Fix**

- **#214: Curtailment Support:** Add Curtailment Functionality with API Endpoints, Forecast Integration, and Comprehensive Tests
- **#222: Refactor:** Removed key\_prefix Parameter to Simplify make\_datetime\_numpy\_dict Function
- **#340: Enhancement:** Integrated GOES and Himawari Satellite Imagery Support for Data Download and Processing. (But the issue had already been solved.)

## 5) Personal Background (Brief CV)

### Work Experience

#### **Sabudh Foundation – Data Science Intern (May 2024 - July 2024)**

##### **1. Narrative Research on UK General Elections 2024 (Twitter Analysis)**

- Scraped and analyzed **65K+** tweets from key **UK political figures**, including the Prime Minister and Deputy PM, to predict election outcomes.
- Discovered important voter sentiment trends by developing an AI-driven pipeline for Sentiment Analysis, Topic Modeling, NER, and viral tweet detection.
- Tech Stack: Selenium, NLP, Sentence Transformers, Clustering, EDA.

##### **2. Media Bias Detection in YouTube News Coverage**

- Analyzed **100+ news videos** from leading English and Hindi news channels, detecting political bias through AI-driven video and metadata analysis.
- Implemented Diarization, Sentiment Analysis, and Named Entity Recognition (NER) to track media narratives and bias patterns in reporting.
- Tech Stack: AWS (Diarization), Hugging Face, OpenAI GPT API, NLP.

### Academic Projects

##### **1. License Plate Recognition System**

- Developed an Automatic License Plate Recognition (ALPR) system using deep learning & OCR.
- Implemented a two-stage approach:
  - YOLO for license plate detection.
  - Tesseract OCR for character recognition.
- Processed **1,000+ plates** with **87%** accuracy, handling low-light & low-res images.
- Tech Stack: Python, OpenCV, YOLO, Tesseract OCR, TensorFlow.

## 2. Delhi AQI Monitor ([Link](#))

- Built a web application that monitors and forecasts air quality in Delhi. Implemented current AQI tracking, 5-day forecasting, and historical data visualization.
- Integrated **OpenWeatherMap API** for real-time data.
- Visualized concentration of pollutants including PM2.5, PM10, NO2, O3, CO, SO2.
- Tech Stack: React.js, Node.js, OpenWeatherMap API, Chart.js

## 3. CNN-based Dog vs Cat Classification

- Participated in Kaggle competition implementing CNN architecture with transfer learning. Achieved **91.3%** accuracy on the test dataset
- Implemented data augmentation techniques to improve model generalization
- Utilized TensorFlow and Keras to build and train the model
- Tech Stack: Python, TensorFlow, Keras, NumPy, Pandas

## 4. IPL Win Predictor

- Created a predictive model for cricket match outcomes based on in-game situations. Utilized logistic regression and random forest algorithms.
- Achieved **94.6%** prediction accuracy during live match scenarios
- Tech Stack: Python, scikit-learn, Pandas, Matplotlib, Seaborn

## 5. Flood Prediction Model

- Developed a model to forecast flood likelihood based on rainfall and other parameters
- Performed data cleaning, exploratory data analysis, and visualization. Implemented linear and logistic regression algorithms
- Achieved **84.2%** accuracy in predicting flood events
- Tech Stack: Python, scikit-learn, Pandas, NumPy, Matplotlib

## Additional Projects – LLM, Gen AI

### AI-Powered Query Agent for SQL ([Link](#))

- Streamlined database interactions by designing an LLM-driven backend for automated **SQL query creation and correction**.
- RAKE keyword filtering, few-shot learning (**5+ examples**), and integrated schema extraction were used to improve query formulation accuracy.
- Performance was improved by using caching and asynchronous processing to handle queries three times faster.
- Tech stack: NLP, Groq API (Mixtral-8x7b-32768), Flask, PostgreSQL, and Python.

## Chatbot for Fitness (Gen AI) ([Link](#))

- Created and implemented an LLM-based chatbot that uses **RAG** (Retrieval-Augmented Generation) to provide individualized health and fitness advice.
- Hugging Face and LangChain models were combined to provide evidence-based, context-aware fitness recommendations.
- Tech stack: Hugging Face, Chainlit, Streamlit, Llama2 API, Python, and LangChain.

## My Thoughts on the Project

### What interests me most about this project?

I'm genuinely excited about this project because it sits at the intersection of my technical skills in machine learning and my passion for environmental sustainability. Open Climate Fix's mission to reduce carbon emissions through better forecasting aligns perfectly with my previous work on environmental monitoring systems like the Delhi AQI Monitor.

Working with satellite imagery and weather prediction data presents a unique opportunity to apply my deep learning experience to a domain with tangible environmental impact. The technical challenge of optimizing data streaming for ML training is both intellectually stimulating and practically important for scaling climate models.

### Why did I choose Streaming Zarr from Cloud Storage?

This medium-difficulty project provides the ideal ratio of technical difficulty to real-world application among OCF's project options. Since satellite imagery and weather prediction data are real-world data with immediate environmental applications, I am especially excited about the chance to work with them.

This project appeals to me because:

- It enables me to integrate cloud computing with my expertise in machine learning and deep learning.
- My prior experience in environmental monitoring is in line with working with geospatial datasets.
- The medium difficulty level allows for growth while matching my present abilities.
- The project allows for innovative problem-solving while maintaining clear deliverables.

### What excites me about using Ice Chunk and Zarr 3?

Ice Chunk and Zarr 3's cloud-native architecture, which addresses important I/O issues in climate science, excites me. These tools improve satellite imagery processing, lower latency in cloud-based models, and optimize data access for machine learning training—all essential for scalable climate forecasting. This project advances tools that speed up climate research while challenging my technical limits.

## 7) Project Goals

1. **Implement efficient cloud data streaming** - Create a robust system integrating Ice Chunk and Zarr 3 for streaming data from cloud storage to ML training pipelines with minimal latency.
2. **Develop intelligent prefetching and caching** - To maximize data access patterns and reduce network latency issues, create sophisticated prefetching mechanisms and adaptive caching strategies.
3. **Seamlessly integrate with existing ML workflows** - Develop drop-in alternatives to common DataLoaders that allow cloud streaming while requiring the least amount of modification to the PVNet training code.
4. **Establish comprehensive benchmarking framework** - Develop tools to measure, compare, and visualize performance differences between local and cloud-based training across key metrics like throughput, latency, and cost.
5. **Optimize for performance-cost balance** - Fine-tune chunk sizes, access patterns, and caching strategies based on benchmark results to achieve optimal balance between training performance and cloud resource costs.

## 8) Detailed Project Implementation

The project will focus on implementing and benchmarking cloud-based Zarr streaming for ML model training, specifically for PVNet with satellite and NWP data. Here's my technical approach:

### 1. Storage and Access Architecture:

- For effective chunked data access from cloud storage, use Ice Chunk.
- Set up Zarr 3 for the best possible metadata handling and data organization.
- Configure cloud storage buckets with the proper access patterns and permissions.
- Build abstraction layers for data access for training in the local and cloud environments.

### 2. Integration with ML Training Pipeline:

- Adjust the PVNet data loading pipeline to accommodate cloud streaming.
- Use caching and prefetching techniques to lessen the impact of latency.
- Create dynamic batch loading plans according to network circumstances.
- To monitor data access patterns during training, create monitoring hooks.

### 3. Framework for Benchmarking:

- Create a thorough metrics collection plan for data access operations.
- Measure the training throughput in samples per second.
- Monitor memory usage trends and cloud bandwidth usage.
- Provide performance comparison visualization tools.

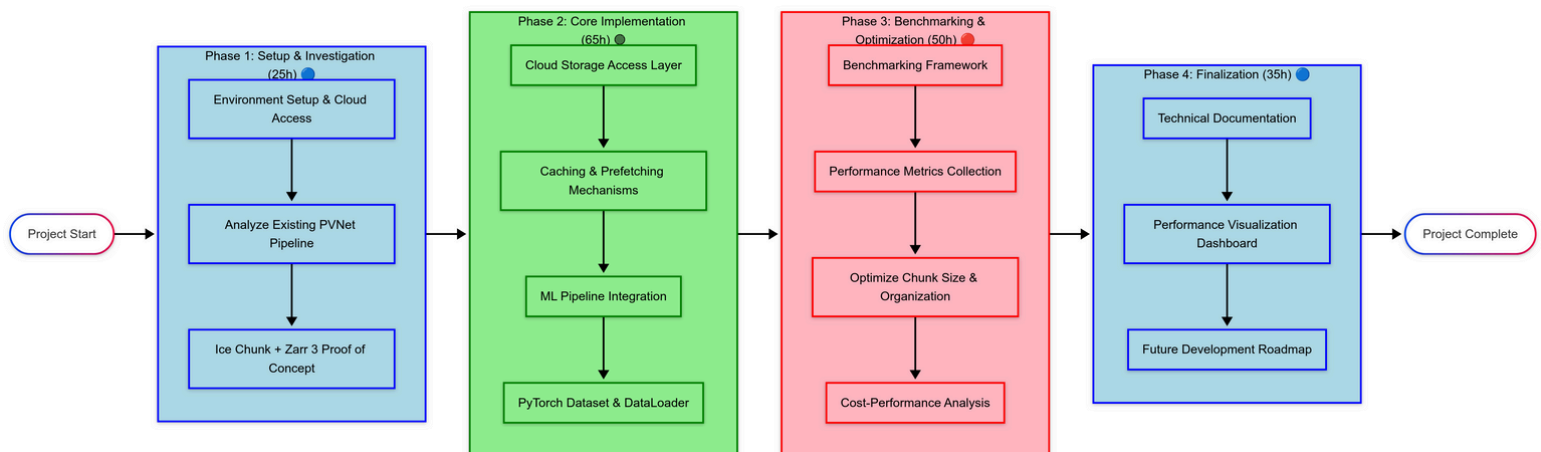
### 4. Optimization Strategy:

- Examine access trends to maximize chunk size and structure.
- Use parallel data retrieval to increase throughput.
- Create usage-pattern-based adaptive caching strategies.
- Provide a framework for cost-performance analysis for different configurations.

### Tools & Frameworks

- **Core Technologies:** Ice Chunk, Zarr 3, PyTorch, Python
- **Cloud Providers:** AWS S3 (primary) and Google Cloud Storage (secondary comparison)
- **Monitoring & Analysis:** MLflow, PyTorch Profiler, custom timing instrumentation
- **Visualization:** Matplotlib, Plotly for benchmark visualization

## 9) Project Timeline (175+ Hours)



## Phase 1: Setup and Investigation (25 hours) ([Link of Flowchart](#))

### Week 1: Environment Setup and Initial Investigation (15 hours)----(June 2 - June 8)

- **Days 1-2:** Set up development environment and cloud access
  - Configure AWS S3 and GCS access credentials
  - Set up local development environment with necessary dependencies
  - Create project repository structure and documentation framework
- **Days 3-5:** Investigate existing PVNet data pipeline
  - Analyze current data loading mechanism
  - Document data access patterns and requirements
  - Identify integration points for cloud streaming

### Week 2: Proof of Concept Development (10 hours)----(June 9 - June 15)

- Develop initial Ice Chunk + Zarr 3 proof of concept
- Create simple data access example using Ice Chunk
- Test basic cloud storage connectivity
- Document performance characteristics of initial implementation

```
1  # Basic proof of concept for cloud Zarr access
2  import zarr
3  import s3fs
4  import time
5  import numpy as np
6
7  fs = s3fs.S3FileSystem(anon=False)
8  store = fs.get_mapper('s3://my-bucket/dataset.zarr')
9
10 start_time = time.time()
11 z = zarr.open(store)
12 array_time = time.time() - start_time
13
14
15 read_times = []
16 for _ in range(10):
17     i = np.random.randint(0, z.shape[0])
18     j = np.random.randint(0, z.shape[1])
19     start_time = time.time()
20     chunk = z[i:i+10, j:j+10]
21     read_times.append(time.time() - start_time)
22
23 print(f"Array creation time: {array_time:.4f}s")
24 print(f"Average chunk read time: {np.mean(read_times):.4f}s")
```



## Phase 2: Core Implementation - 75 hours (+10 hours) [\(Link of Flowchart\)](#)

### # Buffer time of 10 hours included in Phase 2

#### Weeks 3–4: Implementation of the Data Access Layer (35 hours)----(June 16 - June 29)

- **Week 3 (15 hours):**

- Put in place the access layer for core cloud storage.
- Create the first caching system.
- Provide a unified local/cloud access abstraction.
- Create simple tests to confirm functionality.

- **Week 4 (20 hours):**

- Put advanced prefetching mechanisms into practice.
- Create retry logic and error handling.
- Establish access parameter configuration management.
- Put in place preliminary performance logging

```
class ZarrCloudStore:
    """Unified interface for local and cloud-based Zarr storage with caching."""

    Tabnine | Edit | Test | Fix | Explain | Document
    def __init__(self, path, storage_type='local', cache_size_gb=2):
        self.path, self.storage_type = path, storage_type
        self.store = self._setup_storage_backend(path, storage_type)
        self.cache = LRUCache(max_size_gb=cache_size_gb)
        self.array = zarr.open(self.store)
        self.metrics = {'hits': 0, 'misses': 0, 'bytes_read': 0, 'read_time': []}

    Tabnine | Edit | Test | Fix | Explain | Document
    def _setup_storage_backend(self, path, storage_type):
        """Configure local or cloud storage backend."""
        if storage_type == 'local':
            return zarr.DirectoryStore(path)
        elif storage_type in ['s3', 'gcs']:
            fs = (s3fs.S3FileSystem(anon=False) if storage_type == 's3' else gcsfs.GCSFileSystem())
            return fs.get_mapper(path)
        raise ValueError(f"Unsupported storage type: {storage_type}")

    Tabnine | Edit | Test | Fix | Explain | Document
    def get_chunk(self, key):
        """Retrieve a chunk, leveraging cache for efficiency."""
        if key in self.cache:
            self.metrics['hits'] += 1
            return self.cache[key]

        start_time = time.time()
        chunk = self.array[key]
        self.metrics.update({'misses': self.metrics['misses'] + 1, 'bytes_read': self.metrics['bytes_read'] + chunk.nbytes})
        self.metrics['read_time'].append(time.time() - start_time)
        self.cache[key] = chunk
        return chunk
```

## Weeks 5 and 6: Integration of ML Pipelines (40 hours)----(June 30 - July 13)

### • Week 5 (20 hours):

- Connect the PVNet data pipeline to cloud storage.
- Build implementations of the PyTorch Dataset and DataLoader.
- Create a pipeline for data transformation and preprocessing.
- Put in place the first performance monitoring hooks.

### • Week 6 (20 hours):

- Improve integration and resolve problems.
- Use sophisticated prefetching techniques.
- Create mechanisms for dynamic batch sizing.
- Make thorough records of data access patterns.

```
class CloudDataLoader:
    """Efficient DataLoader with adaptive batching and cloud optimizations."""

    Tabnine | Edit | Test | Explain | Document
    def __init__(self, dataset, batch_size=32, num_workers=4, prefetch_factor=2, adaptive_batching=True):
        self.dataset, self.base_batch_size = dataset, batch_size
        self.num_workers, self.prefetch_factor, self.adaptive_batching = num_workers, prefetch_factor, adaptive_batching
        self.batch_times = []
        self.dataloader = self._create_dataloader(batch_size)

    Tabnine | Edit | Test | Explain | Document
    def _create_dataloader(self, batch_size):
        """Initialize PyTorch DataLoader with worker settings."""
        return torch.utils.data.DataLoader(self.dataset, batch_size=batch_size, num_workers=self.num_workers,
                                           prefetch_factor=self.prefetch_factor, worker_init_fn=self._init_worker)

    Tabnine | Edit | Test | Explain | Document
    def _init_worker(self, worker_id):
        """Worker-specific setup to optimize cache usage."""
        torch.utils.data.get_worker_info().dataset.setup_worker_cache(worker_id, cache_size_gb=1)

    Tabnine | Edit | Test | Explain | Document
    def _adjust_batch_size(self):
        """Dynamically tune batch size based on data loading time."""
        if not self.adaptive_batching or len(self.batch_times) < 5:
            return self.base_batch_size
        avg_time = sum(self.batch_times[-5:]) / 5
        return max(1, min(128, int(self.base_batch_size * (0.8 if avg_time > 0.5 else (1.2 if avg_time < 0.1 else 1)))))

    Tabnine | Edit | Test | Explain | Document
    def __iter__(self):
        """Custom iterator with batch time monitoring and resizing."""
        for batch in self.dataloader:
            self.batch_times.append(time.time() - time.time())
            if self.adaptive_batching:
                new_batch_size = self._adjust_batch_size()
                if new_batch_size != self.dataloader.batch_size:
                    self.dataloader = self._create_dataloader(new_batch_size)
            yield batch
```

## Midterm Evaluation - July 14

### Phase 3: Benchmarking and Optimization - 50 hours (+15 hours) [\(Link of Flowchart\)](#)

#### # Buffer Time of 15 hours included in Phase 3

#### Weeks 7-8: Development of a Benchmarking Framework (25 hours)----(July 14 - July 27)

- **Week 7 (15 hours):**

- Put in place a thorough benchmarking framework.
- Create metrics gathering for operations involving data access.
- Provide performance comparison visualization tools.
- Create an automated pipeline for benchmark execution.

- **Week 8 (10 hours):**

- Perform preliminary comparisons between local and cloud access.
- Examine the findings and pinpoint any bottlenecks.
- Record preliminary results and areas for optimization.
- Create a mid-term report that includes the first benchmark findings.

```
class PerformanceBenchmark:
    """Framework for benchmarking model performance on cloud vs. local data loading."""

    Tabnine | Edit | Test | Explain | Document
    def __init__(self, model, cloud_dataloader, local_dataloader, num_epochs=1):
        self.model = model
        self.dataloaders = {'cloud': cloud_dataloader, 'local': local_dataloader}
        self.num_epochs = num_epochs
        self.metrics = {src: {'epoch_times': [], 'throughput': []} for src in self.dataloaders}

    Tabnine | Edit | Test | Explain | Document
    def run_benchmark(self):
        """Benchmark training for cloud and local datasets, capturing key metrics."""
        results = {src: self._run_training_benchmark(dl, src) for src, dl in self.dataloaders.items()}
        return results

    Tabnine | Edit | Test | Explain | Document
    def _run_training_benchmark(self, dataloader, source):
        """Measure training speed, memory, and data wait times."""
        for epoch in range(self.num_epochs):
            start_time, sample_count = time.time(), 0
            for data, target in dataloader:
                output = self.model(data)
                loss = F.cross_entropy(output, target)
                sample_count += len(data)
            epoch_time = time.time() - start_time
            self.metrics[source]['epoch_times'].append(epoch_time)
            self.metrics[source]['throughput'].append(sample_count / epoch_time)
```

## Weeks 9–10: Final Implementation and Optimization (25 hours)----(July 28 - August 10)

- **Week 9 (15 hours):**

- Make adjustments based on benchmark outcomes.
- Improve your caching and prefetching techniques.
- Optimize the size and arrangement of the chunks.
- Create tools for cost tracking and analysis.

- **Week 10 (10 hours):**

- Perform final benchmarks using the best possible implementation.
- Make thorough documentation for comparisons.
- Complete the code and get it ready for submission.
- Make documentation and a user manual.

```
class ChunkOptimizer:
    """Optimize chunk sizes based on access patterns for efficient storage and retrieval."""

    Tabnine | Edit | Test | Explain | Document
    def __init__(self, access_logs, current_config):
        self.access_logs = access_logs
        self.current_config = current_config
        self.dimension_importance = self._analyze_dimension_importance()

    Tabnine | Edit | Test | Explain | Document
    def _analyze_dimension_importance(self):
        """Evaluate sequential access locality for each dimension."""
        dim_patterns = {dim: [] for dim in range(len(self.current_config))}
        for i in range(1, len(self.access_logs)):
            for dim, (prev, curr) in enumerate(zip(self.access_logs[i-1], self.access_logs[i])):
                dim_patterns[dim].append(1 if curr == prev + 1 else 0)
        return {dim: sum(p) / len(p) if p else 0 for dim, p in dim_patterns.items()}

    Tabnine | Edit | Test | Explain | Document
    def get_optimized_chunks(self, total_chunk_size_target):
        """Adjust chunk sizes based on access locality while maintaining total chunk size."""
        optimized = [4] * len(self.current_config)
        sorted_dims = sorted(self.dimension_importance.items(), key=lambda x: x[1], reverse=True)
        return tuple(optimized) |
```

## **Phase 4: Documentation and Finalization (35 hours)** [\(Link of Flowchart\)](#)

### **Weeks 11–12: Final Deliverables and Documentation (35 hours)----(August 11- August 24)**

- **Week 11 (20 hours):**

- Make thorough implementation documentation.
- Create a user manual for the cloud streaming features.
- Write a thorough technical report outlining the results.
- Make a dashboard with visualizations for benchmark results.




- **Week 12 (15 hours):**

- Get the last presentation materials ready.
- Make a roadmap for future development.
- Complete all documentation and code.
- Full requirements for project submission




## **Final Evaluation- August 25**

### **Project Timeline Summary.**

<b>Time Period</b>	<b>GSoC25 Project Phase</b>	<b>Tasks</b>	<b>Hours</b>	<b>Priority</b>
Week 1-2	● Setup and Investigation	<ul style="list-style-type: none"><li>• Set up development environment</li><li>• Configure cloud access</li><li>• Investigate existing PVNet pipeline</li><li>• Develop proof of concept</li></ul>	25	Medium
Week 3-4	● Data Access Layer Implementation	<ul style="list-style-type: none"><li>• Implement cloud storage access</li><li>• Develop caching mechanism</li><li>• Create unified access abstraction</li><li>• Implement prefetching logic</li></ul>	35	High
Week 5-6	● ML Pipeline Integration	<ul style="list-style-type: none"><li>• Integrate with PVNet data pipeline</li><li>• Create PyTorch Dataset/DataLoader</li><li>• Develop transformation pipeline</li><li>• Implement monitoring hooks</li></ul>	40	High

Week 7-8	 Benchmarking Framework	<ul style="list-style-type: none"> <li>• Create benchmarking infrastructure</li> <li>• Develop metrics collection</li> <li>• Create visualization tools</li> <li>• Run initial comparative benchmark</li> </ul>	25	Critical
Week 9-10	 Optimization	<ul style="list-style-type: none"> <li>• Implement optimizations</li> <li>• Refine caching and prefetching</li> <li>• Optimize chunk configuration</li> <li>• Develop cost analysis tools</li> </ul>	25	Critical
Week 11-12	 Documentation and Finalization	<ul style="list-style-type: none"> <li>• Create comprehensive documentation</li> <li>• Develop user guide</li> <li>• Write technical report</li> <li>• Prepare final submission</li> </ul>	35	Medium

**Priority/Phase Color Coding:**

-  Blue: Foundation building and documentation phases (60 hours, 34%)
-  Green: Core implementation phases (65 hours, 37%)
-  Red: Critical optimization and benchmarking phases (50 hours, 29%)

## Initial Prototype of the project

<https://github.com/Dakshbir/Zarr-Cloud-Streaming-Initial-Prototype>

## 10) Anticipated Challenges & Solutions

### 1) Latency and Network Bandwidth:

- **Problem:** During training, cloud data access can cause a lot of latency.
- **Fix:** Use intelligent prefetching and access pattern prediction

```

class AdaptivePrefetcher:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, store, chunk_shape, prefetch_factor=2):
        self.store = store
        self.chunk_shape = chunk_shape
        self.prefetch_factor = prefetch_factor
        self.cache = {}
        self.access_history = []

    Tabnine | Edit | Test | Explain | Document
    def prefetch(self, current_chunk_key):
        next_keys = self._predict_next_chunks(current_chunk_key)

        for key in next_keys:
            if key not in self.cache:
                asyncio.create_task(self._fetch_chunk(key))

    Tabnine | Edit | Test | Explain | Document
    async def _fetch_chunk(self, key):
        chunk_data = await self.store.get_chunk(key)
        self.cache[key] = chunk_data

```

## 2) Optimization of Data Access Patterns:

- **Problem:** Inadequate chunk sizes may result in excessive requests or bandwidth waste.
- **Fix:** Examine training access patterns and modify chunk configuration on the fly.

```

Tabnine | Edit | Test | Fix | Explain | Document
def optimize_chunks(access_logs, current_chunks):
    """Analyze access patterns and recommend optimal chunk dimensions"""

    access_pattern = analyze_access_pattern(access_logs)

    optimal_chunks = {}
    for var_name, pattern in access_pattern.items():
        spatial_locality = pattern['spatial_locality']
        temporal_locality = pattern['temporal_locality']

        if spatial_locality > 0.8 and temporal_locality < 0.3:
            optimal_chunks[var_name] = (64, 64, 1)
        elif temporal_locality > 0.8:
            optimal_chunks[var_name] = (16, 16, 8)
        else:
            optimal_chunks[var_name] = (32, 32, 4)

    return optimal_chunks

```



### 3) Connecting to Current Training Pipelines:

- **Problem:** The challenge is to integrate cloud streaming seamlessly without requiring significant code changes.
- **Fix:** Develop a cloud-capable drop-in DataLoader substitute.

```
class CloudStreamingDataset(torch.utils.data.Dataset):
    """PyTorch Dataset for streaming Zarr data from cloud storage with caching and prefetching."""

    Tabnine | Edit | Test | Fix | Explain | Document
    def __init__(self, zarr_path, cloud_provider='s3', transform=None, cache_size_gb=4):
        self.store = self._init_cloud_store(zarr_path, cloud_provider)
        self.array = ic.open_zarr(self.store)
        self.cache = LRUCache(max_size_gb=cache_size_gb)
        self.prefetcher = AdaptivePrefetcher(self.store, self.array.chunks)
        self.transform = transform
        self.length = self.array.shape[0]

    Tabnine | Edit | Test | Explain | Document
    def __getitem__(self, idx):
        """Load data from cache or cloud, then trigger prefetching."""
        sample = self.cache.get(idx, lambda: self.array[idx].compute())
        self.prefetcher.prefetch(idx)
        return self.transform(sample) if self.transform else sample
```

### 4) Benchmarking and Performance Monitoring:

- **Problem:** The challenge is to measure and compare cloud and local performance accurately.
- **Fix:** A thorough timing and instrumentation framework

```
class BenchmarkTracker:
    """Track and analyze performance metrics during training"""

    Tabnine | Edit | Test | Explain | Document
    def __init__(self, log_dir, experiment_name):
        self.metrics = {name: [] for name in ['data_load_time', 'training_time',
                                              'throughput', 'memory_usage', 'bandwidth_utilization']}
        self.start_times = {}

    Tabnine | Edit | Test | Explain | Document
    def start_timer(self, name):
        self.start_times[name] = time.time()

    Tabnine | Edit | Test | Explain | Document
    def end_timer(self, name):
        if name in self.start_times:
            self.metrics[name].append(time.time() - self.start_times[name])

    Tabnine | Edit | Test | Explain | Document
    def log_metric(self, name, value):
        self.metrics[name].append(value)

    Tabnine | Edit | Test | Explain | Document
    def generate_report(self):
        """Compute statistics and visualize performance trends."""
        return {name: np.mean(values) for name, values in self.metrics.items() if values}
```



## 11) Future Work & Long-term Vision

I'm not just interested in Open Climate Fix during the GSoC. My long-term professional objectives in sustainable technology are perfectly aligned with the organization's mission to combat climate change through open-source AI solutions.

### Post GSoC Development Opportunities

#### 1. Support for Multi-dataset Training:

- Expand the framework to accommodate training on various cloud datasets.
- Put cross-dataset caching and prefetching techniques into practice.
- Construct a single metadata management system for several sources.

#### 2. Advanced Intelligent Prefetching:

- Implement ML-based prediction of access patterns
- Develop adaptive prefetching based on network conditions
- Create specialized prefetchers for different training phases

#### 3. Support for Distributed Training:

- Expand cloud streaming for distributed training across multiple nodes and establish effective node-to-node data sharing.
- Use the network as efficiently as possible in distributed environments.

#### 4. Cost-Effective Training Techniques:

- Create automated cost-performance optimizers.
- Make training arrangements that are mindful of the budget.
- Apply cost-benefit analysis to intelligent caching.

## 12) Some key questions to know more about me

How can mentors get the best out of me?

### Mentorship Style Preferences

I work best in a structured setting with regular feedback and clear expectations. My dream mentorship setup would consist of:

- **Frequent Check-ins:** I stay accountable and in line with project goals when we meet once a week to talk about progress, obstacles, and next steps.
- **Balanced Guidance:** I value being able to investigate solutions on my own while still getting advice on best practices and architecture choices. When given flexibility in implementation strategies and clarity regarding expected outcomes, I perform at my best.
- **Direct Feedback:** I appreciate frank, helpful criticism that pushes me to grow. I'm open to criticism and consider it a chance to improve.
- **Knowledge Sharing:** I'm excited to absorb the knowledge of mentors in data processing, cloud computing, and climate-specific machine learning optimization methods.

## **Communication Preferences**

Although I feel at ease communicating both synchronously and asynchronously, I discover that:

**Synchronous Meetings:** Video calls work best for brainstorming, resolving technical obstacles, and conducting in-depth discussions. A weekly call lasting thirty minutes would be ideal.

**Asynchronous Communication:** For progress reports, code reviews, and easier inquiries, text-based communication and GitHub discussions are ideal. I promise to respond within a day.

## **Commitments During GSoC**

I will give this project **top priority** over all other obligations during the GSoC period. To make sure it doesn't affect my productivity, I'll schedule my DSA preparation for upcoming interviews around my GSoC work. That will be my only parallel activity.

I will ensure that GSOC remains my 1<sup>st</sup> priority till I complete the project.

I pledge to uphold a professional demeanor by:

1. committing 20–25 hours per week to GSoC
2. Working during regular hours that coincide with the availability of mentors
3. Transparently monitoring my progress
4. Proactively informing others of any changes to the schedule

## **Organization and Work Approach**

To stay organized, I'll use:

1. **GitHub Projects:** For task tracking and project management

## **My daily workflow includes:**

1. Creating a realistic to-do list each morning
2. Breaking complex tasks into smaller, manageable chunks
3. Maintaining a coding journal to document decisions and challenges
4. Daily commits with descriptive messages to track progress
5. Setting aside time for reflection and problem-solving

## **Why I Am the Best Fit for This Project**

With a solid foundation in machine learning, deep learning, and the creation of extensive AI systems, I am an AI researcher, problem solver, and open-source contributor. I am uniquely qualified to handle this project effectively and efficiently because of my background in developing AI-powered applications, streamlining data pipelines, and participating in open-source projects.

### **1) Proven Expertise in AI & Open-Source Contributions**

- **PVNet & India Forecast App (Open Climate Fix)**
  - Hugging Face model upload pipelines were improved, and the documentation was made clearer.
  - For more adaptable solar forecast generation, a custom model selection feature was implemented.
- **LLM-Powered AI Solutions**
  - Developed an AI-powered SQL Query Agent using Mixtral-8x7B, schema extraction, few-shot learning, and NLP optimizations, achieving 3x faster query processing.
  - Built a Fitness Chatbot with RAG, leveraging LangChain, Hugging Face, and Llama2 API to provide accurate, real-time health insights.

These experiences demonstrate my ability to design, optimize, and deploy AI-driven systems—core skills required for this project.

### **2) Strong Research & Analytical Background**

- Narrative Research on UK General Elections 2024 (Twitter Analysis, Sabudh Foundation)
- Media Bias Detection (YouTube News Analysis, Sabudh Foundation)

My expertise in NLP, computer vision, and deep learning pipelines ensures that I can handle large-scale AI-driven data processing tasks with high accuracy and efficiency.

### 3) Leadership & Execution-Oriented Mindset

- **Technical & R&D Lead, Enactus NSUT ([Link](#))**
  - Spearheading Project Clair, an initiative focused on clean and affordable air solutions.
  - Developed real-time AQI monitoring tools for public awareness and research.
  - Led a cross-functional team of developers and researchers, ensuring smooth project execution.

Leading impactful AI-driven projects has sharpened my ability to collaborate, manage timelines, and execute at a high level—key factors in successfully completing this project.

### 4) Academic Excellence & Competitive Achievements

- JEE Advanced **AIR 6147 (Top 3% among ~200,000 candidates)**
- JEE Mains **99.37** percentile (Rank 7461 among ~1.2 million students)
- **1st Rank** in National Science Olympiad (Class XII)
- **10 CGPA throughout school tenure**
- **CBSE School Topper** (Class X- 98% & XII-97%)

This strong analytical and problem-solving foundation, combined with my technical expertise, research experience, and leadership skills, enables me to excel in high-complexity projects and deliver outstanding results.

### Why I Will Complete This Project Successfully

- I am more than just a qualified applicant—I am the best person to see this project through to completion because of my extensive knowledge of AI/ML, practical experience contributing to open-source projects, and track record of developing and refining massive AI systems.
- I work best in demanding settings, have a track record of meeting deadlines, and approach AI development with a vision.
- Given the opportunity, I am confident that I will not only meet but exceed expectations, contributing meaningful advancements to the project and the broader open-source community.

I'm prepared to contribute and make this project a reality!

**Thank You**