GSoC 2025 Proposal – Apache Beam: ML Vector DB/Feature Store Integrations (Pinecone & Tecton Connectors)

Personal Introduction & Information

Name : Wesam Abed
Email : wesamwaleed22@gmail.com
Phone : +972595065849
Address : Israel WestBank Ramallah
Linkedin : www.linkedin.com/in/wesam-abed-412a25243

Wesam Abed is a motivated software engineer with a Bachelor's degree in Computer Engineering from Birzeit University. I have professional experience developing backend systems at **525K**, **MenaForce**, and **Asal Tech**, where I honed my skills in building scalable, distributed applications. Over the years, I have specialized in backend development with a focus on **distributed systems** and **scalable architecture**.

My technical expertise includes:

- **Programming Languages & Frameworks:** Proficient in Java and Python, with extensive experience in building web services and data pipelines.
- **Databases & Search Engines:** Hands-on experience with Elasticsearch, MongoDB, Amazon DynamoDB, and PostgreSQL, enabling me to design efficient data storage and retrieval solutions.
- **Cloud & DevOps:** Proficient with AWS cloud services and containerization/orchestration tools such as Kubernetes and Docker, ensuring that solutions are cloud-ready and easily deployable.

I have applied these skills in complex projects, including an **AI-driven fraud detection system**, a **multilingual chatbot platform**, and a **drone delivery simulation**. In the fraud detection project, I worked on real-time data ingestion and anomaly detection pipelines; for the chatbot, I integrated NLP models and managed multi-language support; and the drone simulation required coordinating real-time events in a distributed environment. These projects demanded robust and scalable back-end architectures, reinforcing my ability to deliver high-performance solutions. I am fully comfortable working in Python (which is ideal for this Apache Beam project) and equally

adept in Java, which will help in understanding Beam's multi-language environment. With this background, I am well-prepared to contribute to Apache Beam by implementing new I/O connectors for cutting-edge ML infrastructure.

Project Abstract

Project Title: *Apache Beam Connectors for Pinecone Vector DB and Tecton Feature Store*

This project aims to enhance Apache Beam's machine learning pipeline capabilities by developing **new I/O connectors (source and sink)** for **Pinecone** and **Tecton**. Pinecone is a managed vector database service for high-dimensional embeddings, and Tecton is a feature store platform for ML features. Currently, Apache Beam's Python SDK lacks native connectors for these systems, which are increasingly vital in ML workflows like feature engineering and retrieval-augmented generation. The proposed work will implement Beam transforms to **read from and write to Pinecone and Tecton**, enabling seamless integration of vector similarity search and feature store operations into Beam pipelines. Over the standard 12-week timeline, I will deliver production-quality connectors (in Python), along with thorough tests, documentation, and performance benchmarks. By the end of the project, Apache Beam users will be able to easily incorporate Pinecone and Tecton into their batch or streaming pipelines, unlocking new use cases in AI and ML on a unified, scalable data processing framework.

Problem Statement and Motivation

Apache Beam is a powerful unified model for defining data processing pipelines, widely used for batch and streaming workloads. In machine learning use-cases, Beam can be employed for tasks like feature generation, embedding computation, and data preparation. However, as of early 2025, **Beam integrates with only a limited set of feature stores and vector databases**, which constrains its utility in advanced ML pipelines

issues.apache.org
. Critical ML infrastructure components such as vector databases and feature stores are not yet supported with out-of-the-box connectors. This gap forces developers to either create custom pipeline stages outside of Beam or avoid Beam for those portions of the pipeline, resulting in fragmented systems.

Two prominent technologies illustrate this gap: **Pinecone** and **Tecton**. *Pinecone* is a fully managed **vector database** platform purpose-built to handle high-dimensional data (vector embeddings) with advanced indexing and similarity search capabilities

datacamp.com
. Vector databases like Pinecone allow machine learning applications to store embeddings and perform fast nearest-neighbor queries for tasks such as semantic search and recommendation. *Tecton*, on the other hand, is a **feature store** that manages, stores, and serves machine learning features in a scalable and reliable way

getindata.com
. Feature stores like Tecton provide a central repository for engineered features used during model training and online inference, ensuring consistency and low-latency access to feature values.

These systems are increasingly essential for modern AI applications: Pinecone enables retrieval-augmented generation (RAG) and similarity search over embeddings, while Tecton streamlines feature engineering and online feature serving for ML models. Without native Beam connectors, integrating Beam with a vector DB or feature store requires ad-hoc solutions. **The motivation** for this project is to **empower Apache Beam users to incorporate Pinecone and Tecton directly into their data pipelines**. By providing official Beam I/O connectors, we can make Beam a one-stop solution for complex ML data workflows. This will enable use cases such as:

- **Feature Engineering Pipelines:** Compute features in Beam and directly **write to Tecton** for serving, or **read from Tecton** to gather training data in Beam.

- **Embedding Pipelines for RAG:** Generate embeddings in a Beam pipeline and **upsert them into Pinecone**, or **query Pinecone** from Beam to fetch relevant vectors for downstream processing.

- **Unified Batch and Streaming Workflows:** Use Beam's unified model to handle both batch feature backfills and streaming updates to the feature store or vector DB, using the same connectors.

By addressing the current lack of connectors, this project will **bridge Apache Beam with state-of-the-art ML data stores**, significantly expanding Beam's capabilities in AI/ML scenarios

issues.apache.org
. It aligns with the Apache Beam community's goal of building a "rich ecosystem of connectors" for ML applications
issues.apache.org
, and it will help Apache Beam stay relevant as an ML data processing framework. This integration will save developers time (no need to write custom integration code) and ensure that performance and scalability are handled in a consistent, community-vetted manner.

Proposed Deliverables

By the end of the GSoC 2025 coding period, the following deliverables will be completed:

- **Pinecone Connector (Source & Sink):** Implementation of Apache Beam transforms to interact with Pinecone:

    - *WriteToPinecone* PTransform: A sink that takes a PCollection of vector data (embeddings with IDs and metadata) and upserts them into a Pinecone index. It will utilize Pinecone's Python client or REST API for efficient batch upsert operations.

    - *ReadFromPinecone* PTransform: A source that allows reading data from Pinecone. This may support use cases such as fetching vectors by IDs or querying by an embedding (e.g., performing a similarity search for each input query vector). The design will consider how to represent query results (nearest neighbors) as a PCollection.

- **Tecton Connector (Source & Sink):** Implementation of Beam transforms for Tecton:

    - *ReadFromTecton* PTransform: Allows retrieval of feature values from Tecton's feature store. For example, given a PCollection of entity IDs or timestamps, this transform will call Tecton's API (using the Tecton Python client or HTTP API) to fetch the corresponding feature vectors or feature values. This is useful for creating training datasets or enriching data within a pipeline.

    - *WriteToTecton* PTransform: A sink transform to write or push computed features into Tecton. This will use Tecton's provided interfaces to ingest batch feature data or update the online store. If Tecton offers a batch ingestion API or offline store update path, the connector will leverage that.

- **Integration Tests & Example Pipelines:** A suite of tests to verify the correctness of each connector:

    - Unit tests with mocked Pinecone/Tecton clients to simulate API responses and ensure edge cases (empty collections, error handling, etc.) are handled.

    - Integration tests that run the Beam connectors against a real or simulated Pinecone index and Tecton environment (using test API keys or local instances if available) to ensure end-to-end functionality.

    - Example pipeline code in Beam's examples directory (or documentation) demonstrating typical usage of the new connectors (e.g., a pipeline that reads from Tecton, does some processing, and writes to Pinecone).

- **Documentation:** Comprehensive documentation for each connector:

    - User guide pages on the Apache Beam website documenting how to use `ReadFromPinecone`, `WriteToPinecone`, `ReadFromTecton`, and `WriteToTecton` (including code snippets and explanation of configurations,

such as connection parameters, authentication, batch sizing, etc.).

    - In-line code documentation (docstrings and comments) following Beam's contribution guidelines.

    - A design document (if required by the Beam community) detailing the approach and API of the connectors, to be shared on the Beam dev mailing list for feedback.

- **Performance Benchmark Report:** An evaluation of the connectors' performance and scalability. This will include measurements of throughput and latency for reading/writing to Pinecone and Tecton under various conditions (e.g., batch sizes, number of concurrent workers) and suggestions for tuning. The results will ensure that the connectors meet practical performance requirements for typical ML pipeline sizes.

- **Contribution to Beam Ecosystem:** All code will be contributed via Apache Beam's repository (as pull requests). The ultimate deliverable is the **merging of the Pinecone and Tecton connectors into Apache Beam's codebase**, making them available in a future Beam release. Success here includes adhering to the community's code standards, passing code reviews, and receiving approvals from project committers.

Each deliverable will be developed in alignment with Beam's standards for I/O connectors (e.g., respecting the Beam I/O APIs and patterns). By producing both source and sink transforms for each system, we ensure that Beam pipelines can **both ingest data from and output data to Pinecone and Tecton** seamlessly. This dual capability (read/write) is crucial for closed-loop pipelines (for instance, generating embeddings and then using them for retrieval within Beam, or reading features, transforming them, and writing new features back).

Technical Approach and Architecture

Overall Design: The connectors will be implemented in **Apache Beam's Python SDK**, as Python is well-suited for integrating with the available client libraries for Pinecone and Tecton. Given my proficiency in Python, I will focus on writing idiomatic, efficient Python transforms. The design will follow Apache Beam's established patterns for I/O connectors, which typically involve implementing `PTransform` classes with underlying `DoFn` or `Impulse`/`Read` sources for reading, and `DoFn`/`Write` operations for sinks. Thread-safety, checkpointing (for streaming), and efficient I/O will be key considerations.

Pinecone Connector Architecture:

- *WriteToPinecone:* I will implement this as a `PTransform` (likely using `ParDo` internally) that takes in a `PCollection` of items (each item containing an ID, vector, and optional metadata) and writes them to a Pinecone index. The connector will use the **Pinecone**

Python client for ease of integration. For performance, the writing will be batched: Beam's `GroupIntoBatches` can be used to batch a set of items, then a DoFn will upsert a batch of vectors in one API call (to amortize network overhead). The transform will handle **authentication** (using an API key and environment configuration for Pinecone – these can be provided via `PipelineOptions` or transform parameters). It will also include error handling and retry logic (leveraging Beam's retry decorators or manual try-except) to gracefully handle transient network issues or rate limits. If Pinecone returns failures for specific items, the connector can either retry or produce meaningful error logs/metrics. The design will ensure **idempotency** where possible – for example, upserting vectors is naturally idempotent in Pinecone (same ID will overwrite), which is good for exactly-once semantics in Beam if needed.

- *ReadFromPinecone:* Reading from Pinecone can be approached in a couple of ways:

    1. **Parallel Queries:** If the use-case is to perform similarity searches (e.g., given a query vector or set of query vectors), the connector can be a simple `ParDo` that, for each input query, calls Pinecone's query API and emits the results (nearest neighbor IDs and scores). This would allow a pipeline stage like `pcoll | "QueryPinecone" >> ReadFromPinecone(query_fn=...)`. In this mode, the transform acts more like a map operation using Pinecone's service. This could be implemented as a DoFn that calls `index.query(...)` for each element.

    2. **Full Dataset Read:** If needed, the connector could support reading all vectors from a Pinecone index (for example, to migrate or analyze them). Pinecone's API provides methods to fetch vectors by ID or list all IDs. We can implement a Beam *UnboundedSource* or *BoundedSource* that enumerates IDs (possibly using pagination if Pinecone supports listing in pages) and reads all data. This would be a more complex source transform but useful for batch pipelines. Initially, the project will focus on the first approach (parallel queries or reads by keys) since it covers common ML scenarios; a full-index scan feature can be added if time permits.

- *Data Structures and Output:* The Pinecone query results (for a search) would likely be a list of nearest neighbors (ID and similarity score, plus maybe metadata). We'll define a Python object or a Beam Row to represent this (e.g., an object with fields: query_id, result_id, score, metadata). For the sink, the input might be a tuple (id, vector, metadata) or a small custom class; the transform will be documented to specify the expected input format.

Tecton Connector Architecture:

- *ReadFromTecton:* Tecton usage typically involves retrieving feature values for given entities or events. Using the **Tecton Python client** (or their REST API), we can fetch features from the online feature store. The `ReadFromTecton` PTransform might accept a PCollection of "feature requests" (each could be a key or a composite key like (entity_id, timestamp) depending on whether we want online or offline retrieval). For each request, a DoFn will call Tecton's `get_features()` method (or similar) to retrieve the feature vector. The result (a set of feature name-value pairs) will be output as a dictionary or a Beam Row with fields for each feature. This connector will manage **authentication** via API keys and endpoint URLs (provided by the user as parameters). Just like with Pinecone, batching can improve throughput: if the API supports fetching multiple entities in one call, the connector will batch requests (e.g., group a set of entity IDs and retrieve features for all in one API call). This would significantly improve performance for batch pipelines.

- *WriteToTecton:* Writing to a feature store might involve calling an ingestion API or writing to a particular offline store that Tecton monitors. I will research Tecton's capabilities for ingesting data. If Tecton offers a direct API to push new feature values (for example, to the offline store or to an online store via an HTTP endpoint), the connector will use that. Possibly, Tecton could ingest data from a stream or file – if so, the Beam sink could prepare data in the required format. For instance, if Tecton uses Kafka or Kinesis for streaming ingestion, Beam could write to that pipeline (though that might be out of scope; we likely assume a direct API or library method exists). The WriteToTecton transform will take a PCollection of feature data (perhaps key-value maps of feature names to values, along with entity IDs and timestamps) and send them to Tecton. Ensuring exactly-once or idempotent writes will be important (maybe using upsert semantics if available, or careful coordination if necessary). As with Pinecone, errors from Tecton's API will be handled with retries and clear logging.

- *Schema Considerations:* Feature stores like Tecton have predefined feature definitions. The connector will need to interface with features that are already defined in Tecton. The user might have to supply the feature service or feature table name that they want to read from/write to. The transforms will be designed with parameters for things like Tecton workspace, feature service name, etc. Internally, the connector will map the Beam data to Tecton's data model (for example, constructing the request payload for `get_features` or `push_features` accordingly). I will ensure the connector is flexible enough to handle typical feature schemas, possibly by allowing the user to pass a schema or letting Tecton's client return data in a generic structure.

Common Architectural Considerations:

- **Scalability:** Both connectors will be designed to run in a distributed manner. Beam will parallelize the operations across workers. For Pinecone queries or Tecton fetches, each worker can handle a subset of queries in parallel. If the Python SDK's runtime

environment supports multithreading or async, we may utilize async API calls or thread pools to further increase throughput (for example, using Python's `asyncio` or concurrent futures in a DoFn to allow multiple outstanding requests). Beam's model also allows us to use a **RestrictiveSplittableDoFn** for reading if needed, but given the nature of these APIs (external service calls), simple parallel calls might suffice.

- **Latency vs Throughput:** We will test different batch sizes for writes and perhaps for read (if multiple fetch in one call) to find a good balance. The connectors might expose tuning knobs (e.g., a parameter for batch size or query concurrency) so that users can adjust based on their use case (small real-time queries vs. large batch sync).

- **Fault Tolerance:** Beam's fault-tolerance (retries on failure, possible reprocessing on worker restart) will be taken into account. Upserts to Pinecone are idempotent by design (same ID can be written again). For Tecton, if repeated writes could duplicate data, we will document that the pipeline may need deduplication logic unless the Tecton API handles it. The connectors will integrate with Beam's checkpointing/watermark mechanics if in streaming mode (for instance, if reading from Tecton's online store in a streaming loop, we might not use it that way but if so, ensure correct watermark progression).

- **Security:** Both Pinecone and Tecton require authentication (API keys or tokens). The connectors will not hard-code any sensitive info; instead, they will accept credentials via configuration (PipelineOptions or environment variables that Beam can pick up). We will use Beam's recommended practices for handling secrets (possibly using `apache_beam.io.filesystems` if needed to read from a secure file, or just direct options).

- **Architecture Diagram:** (If this were a longer document, an architecture diagram would illustrate how data flows from Beam to Pinecone/Tecton and back, but in text: The data flows from PCollection -> connector transform -> external service (Pinecone/Tecton) -> back to PCollection for sources. Each connector acts as a bridge between Beam's pipeline and the external system's API.)

Given my strong background in distributed systems and scalable architecture, I will apply those principles here. For example, my experience with **Elasticsearch** (another distributed data store) gives me insight into indexing and querying performance, which is analogous to Pinecone's usage for vectors. I will ensure the connectors use bulk operations and parallelism effectively, similar to how one would efficiently index documents in Elasticsearch or fetch batches of results. Additionally, my familiarity with **AWS and cloud infrastructure** means I understand the network and deployment environments in which these connectors will run (possibly on Dataflow or Flink runners in the cloud), so I will design with high-latency networks and large scale in mind.

Testing and Benchmarking Strategy

Testing is crucial for connectors that interact with external systems. I plan a multi-tier testing approach:

- **Unit Testing with Mocks:** For each connector, I will write unit tests that use **mock objects** for the Pinecone and Tecton clients. For example, using Python's `unittest.mock`, I can simulate the Pinecone index's `upsert` and `query` methods to return expected responses. Similarly, mock the Tecton client's `get_features` or ingestion calls. These tests will cover:
    - Basic success path: e.g., writing a small batch of vectors yields the expected API calls.
    - Error handling: simulate API exceptions or partial failures and verify the connector retries or handles the error as designed.
    - Edge cases: empty PCollection input (should result in no calls but no errors), extremely large vectors or feature values (ensure no serialization issues in Beam), invalid user parameters (e.g., missing index name or feature service name should raise a meaningful error).
- **Integration Testing:** I will set up integration tests that run the actual connector code against a real instance of the services:
    - For **Pinecone**, I can use a Pinecone trial or free tier to create a test index. The integration test (likely marked as ignored or optional in CI due to external dependency) would configure the Pinecone API key and environment, then run a Beam pipeline that writes a few vectors and reads them back. We will verify that the data round-trips correctly (i.e., what's written can be read via query or fetch). If running as part of automated tests is not feasible (due to external calls), I will at least perform these tests manually and include the results and perhaps provide a way for maintainers to run them.
    - For **Tecton**, since it's a commercial feature store, full integration testing is trickier. I will explore if Tecton offers a sandbox or open-source variant (Tecton was built by creators of Uber's Michelangelo; an open alternative is **Feast**, but we focus on Tecton specifically). If a Tecton trial environment is available, I will use it to test reading and writing of dummy features. Alternatively, I might stub the HTTP API: for example, run a local server that mimics the Tecton API endpoints (this could be done with Flask or a tool like responses library to intercept HTTP calls). The goal is to ensure our connector's logic is correct against something that resembles the real system.

    - Additionally, I will test the connectors in a real Beam runner (not just DirectRunner). Using Google Dataflow or another runner with the connectors would ensure there are no serialization or runtime issues in distributed execution. This might be done late in the project when connectors are ready (e.g., deploy a pipeline on Dataflow that writes to Pinecone).

- **Performance Benchmarking:** To measure performance, I will create test pipelines that push the connectors to handle larger loads:

    - For Pinecone: measure throughput by writing, say, 10k vectors of dimension 100 and see how many per second can be ingested with batch size N. Vary N (maybe 50, 100, 1000) and measure. Similarly, measure query latency for single vs batched queries (if applicable).

    - For Tecton: measure how many feature fetches per second the connector can do when grouping (e.g., fetch 100 entities per call vs 1 per call).

    - Use Beam's **Metrics** API inside the DoFns to record timing (e.g., how long each external API call takes, count of records processed) and aggregate those after pipeline completion.

    - These benchmarks will not only validate performance but also guide default settings (for example, if batch size 100 yields 5x throughput of batch size 1, we might default to 100).

    - I will document the environment of tests (machine specs, network) to contextualize the numbers.

- **Quality Assurance:** Besides automated tests, I will perform code reviews with mentors and possibly get community members to try out the connectors on sample data. Early in development, I plan to write a small demo pipeline (perhaps a dummy "Hello Pinecone/Tecton" pipeline) to manually run and verify behavior. This iterative testing will catch issues like resource leaks (ensuring clients are properly closed if needed) or corner-case exceptions.

- **Benchmark Success Criteria:** The connectors should be able to handle at least on the order of **several thousand records per second** in a typical environment (for writes), without significant degradation, and scale linearly with more workers. Latency for individual read queries should ideally remain in the low hundreds of milliseconds (mostly dependent on external service). If any performance issue arises (for instance, if using Python introduces GIL contention), I'll investigate optimizations (like releasing GIL in I/O waits, or using asynchronous requests).

- **Testing Documentation:** I will include documentation on how to run the tests, especially if credentials are needed for integration tests (with warnings to not commit secrets). The aim is that future maintainers can easily run and extend these tests when Pinecone or Tecton versions update.

By following this testing strategy, we ensure that the connectors are **robust, reliable, and performant**. My experience in back-end and cloud environments has taught me the importance of testing under real-world conditions; for example, in the AI-fraud project, we extensively tested the pipeline with both typical and extreme data to ensure consistency. I will bring the same rigor to this project.

Success Metrics

To evaluate the success of this GSoC project, I will use the following metrics and criteria:

- **Feature Completion:** All planned features (Pinecone source/sink and Tecton source/sink) are implemented and functional. Success means a user can use these connectors to accomplish real tasks (e.g., write a Beam pipeline that successfully writes to Pinecone and one that reads from Tecton).

- **Code Integration and Quality:** The code meets Apache Beam's contribution standards and is merged into the Beam repository. A key metric is passing the code reviews from mentors and the Beam community. Successful integration would be marked by the closure of the JIRA issue (Beam JIRA GSOC-279) and the connectors being included in the Beam codebase.

- **Testing Coverage:** Achieve high test coverage (aiming for >90% of the new code). All unit tests and integration tests should pass. Additionally, no regressions or breaking of other Beam features (ensured by Beam's existing test suite) – essentially a green build.

- **Performance Benchmarks:** The connectors should meet baseline performance requirements. For instance, a metric might be: *The Pinecone sink can ingest at least 5,000 vectors/second on a single worker with batch size 100* (example figure; actual results will vary by environment). If the connectors significantly underperform common expectations, that would need addressing. Ideally, performance tests should demonstrate that the connector throughput scales with parallelism and that overhead introduced by Beam is minimal (within, say, 10-15% of what a raw client could do in a simple script).

- **Documentation & Usability:** Documentation completeness is a metric: the presence of clear how-to guides and examples. Success is when a new user (e.g., a Beam contributor or mentor) can follow the docs to use the connector without needing to read the source. If during evaluation mentors can run the example and it works as

documented, that's a positive sign. Also, API simplicity can be a qualitative metric – for example, the `ReadFromPinecone` transform ideally should be as easy to use as existing connectors (like `ReadFromMongoDB` etc.). Feedback from the community (on the mailing list or PR comments) about the API design being user-friendly would indicate success.

- **Use-Case Validation:** As a measure of the project's value, I plan to demonstrate a realistic use-case pipeline that ties everything together (for instance, a mini pipeline that takes some text data, generates embeddings, writes them to Pinecone, then does a query from Pinecone to find similar items). If this end-to-end demo runs correctly and showcases the connectors working in concert, it's a strong validation of success. The quality of this example (in terms of correctness and performance) is a metric as well.

- **Community Approval:** While harder to quantify, an important success factor is positive engagement from the Apache Beam community. If by the end of the project, project mentors and community members express satisfaction (through comments or acceptance of the work), that indicates we met the expectations. In GSoC terms, a successful final evaluation from the mentors is the ultimate metric.

- **Future Maintainability:** Another success criterion is that the connectors are maintainable and extendable. This can be measured by the clarity of code (for instance, mentor comments like "the code is well-structured" or no significant refactor requests). Also, if follow-up issues or improvements are identified and can be easily added (maybe someone suggests adding another method, and it's straightforward), that means the foundation is solid.

By these metrics, I expect the project to not only deliver the promised features but also to integrate smoothly into Apache Beam's ecosystem, thereby ensuring long-term success beyond the GSoC period.

Timeline

Below is a **12-week timeline** outlining the key phases and tasks of the project. (Assuming a start of the coding period in early June 2025 for GSoC and running 12 weeks through August 2025.)

Week	Milestones & Deliverables

Week 1 (June 1–7)	**Community Bonding & Planning:** Finalize understanding of Pinecone and Tecton APIs. Engage with mentors and Apache Beam dev community to discuss the design approach. Set up development environment (Beam codebase, accounts for Pinecone/Tecton). Prepare a brief design draft for connectors and get initial feedback.
Week 2 (June 8–14)	**Pinecone Connector – Setup & Initial Coding:** Start implementing the Pinecone sink (WriteToPinecone). Define data model for inputs. Write basic version of the DoFn that calls Pinecone's upsert. Also, implement authentication handling (e.g., reading API key from options). Begin a simple unit test for this functionality.
Week 3 (June 15–21)	**Pinecone Sink Completion:** Complete the WriteToPinecone transform with batching and retry logic. Write comprehensive unit tests (mocking Pinecone client). Simultaneously, start implementing ReadFromPinecone for basic query-by-ID functionality. By end of week, have a demo pipeline that writes sample vectors to Pinecone (possibly manual test with a real Pinecone instance).
Week 4 (June 22–28)	**Pinecone Source (Read) Implementation:** Expand ReadFromPinecone to support vector similarity queries. Implement it as a ParDo that takes a PCollection of query vectors and returns nearest neighbor results. Write unit tests for the reading logic (mocking query responses). Conduct an integration test of the full Pinecone connector: pipeline that writes then reads. Address any issues found.
Week 5 (June 29–July 5)	**Documentation & Polish for Pinecone:** Write user-facing documentation for Pinecone connectors (usage examples, parameters). Refine error handling and edge cases based on feedback. If Pinecone connector development finishes early, begin exploratory coding for Tecton (especially understanding how to write to Tecton). Prepare for mid-term evaluation: ensure Pinecone connectors are in a solid state.
Week 6 (July 6–12)	**Mid-term Evaluation & Buffer:** This week marks roughly the mid-term. Complete any remaining Pinecone tasks: integration test results documented, performance tuning (maybe try different batch sizes). Submit Pinecone connector code for initial review by mentors. Receive mid-term evaluation feedback. If time permits, set up the Tecton environment (API keys, define a dummy feature in Tecton for testing).
Week 7 (July 13–19)	**Tecton Connector – Read Implementation:** Begin coding ReadFromTecton. Connect to Tecton's API (or simulate if needed). Implement retrieval of features for a batch of entity keys. Focus on the offline/batch reading scenario first (since online inference read is similar). Write unit tests using a mock Tecton client (simulate feature data returned). Ensure the transform can output data in a structured format (e.g., dictionary of features per entity).

Week 8 (July 20–26)	**Tecton Connector – Write Implementation:** Implement WriteToTecton to push data to the feature store. This might involve formatting data and calling an ingestion API. Write tests for the write path (mocking the API response, ensuring data formatting is correct). If parts of Tecton integration are unclear (due to lack of full access), collaborate with mentors or consider fallback (maybe integrate with Feast as a proxy, but since proposal is specifically Pinecone/Tecton, focus on Tecton best-effort). By end of week, basic Tecton read & write code is written.
Week 9 (July 27–Aug 2)	**Integration Testing & Refinement for Tecton:** Run an integration test for Tecton connector. Possibly use a trial Tecton account to test reading/writing a sample feature. Debug any issues (authentication, data formatting). Fine-tune error handling and add retry logic similar to Pinecone. Also, update documentation for Tecton usage (list required parameters like feature store name, how to supply API key). Start writing a usage example for Tecton connector (perhaps how to use it to build a training dataset in Beam).
Week 10 (Aug 3–9)	**Performance Testing & Optimization:** Conduct the planned benchmarking experiments for both Pinecone and Tecton connectors. Profile the code for any bottlenecks (for example, ensure that no unnecessary serialization happens per element). Optimize batch sizes or parallelism settings if needed. Collect metrics and compile a brief report of results. During this week, also seek **community feedback** by sharing the design and any code snippets on the Beam dev mailing list – incorporate any late feedback or suggestions.
Week 11 (Aug 10–16)	**Final Documentation and Cleanup:** Complete all documentation: ensure the Beam website pages for these connectors are ready (in the required markdown format in Beam's repo). Double-check that all public classes and functions have clear docstrings. Clean up the code (remove debug logs, ensure naming consistency). Prepare the final pull request(s) encompassing the entire feature. This week, I will also devote time to help the Beam community test the connectors if anyone is available (maybe a mentor can try out the PR).
Week 12 (Aug 17–23)	**Final Submission and Wrap-up:** Address any final review comments from the Beam committers on the pull requests. Merge the connectors into the Apache Beam codebase (assuming approvals). Write a **summary report** of the project (and perhaps a blog post on my personal blog or Beam blog if appropriate) describing the journey and outcomes. In the last days, ensure that the connectors are listed in Beam's I/O catalog and that JIRA issue GSOC-279 is marked resolved. Submit final work for GSoC evaluation.

Note: The above timeline is somewhat optimistic but includes buffer in Week 6 (mid-term) and Week 12 for unforeseen delays. If any task takes longer, I will adjust subsequent weeks (for example, if Tecton integration proves complex, some benchmarking might be simplified or done after the official period as additional work). However, the goal is to have Pinecone integration

finished by the mid-term and Tecton by the final evaluation. I will maintain flexibility and constant communication with mentors to re-prioritize if needed.

Throughout the timeline, I will maintain **weekly updates** on my progress (e.g., on the mailing list or project tracker) to keep the community informed and to receive continuous feedback.

Community Engagement Plan

As an aspiring open-source contributor, I plan to actively engage with the **Apache Beam community** and my project mentors to ensure the success of this project and alignment with the community's needs. My engagement plan includes:

- **Apache Beam Dev Mailing List:** I will subscribe and introduce myself on the Beam developer mailing list at the start of the project. I will use the mailing list to discuss design decisions and ask for feedback on proposed approaches. For example, I intend to send a design overview of the Pinecone and Tecton connectors early in the project for community input. I will also post weekly progress summaries to the list (or the appropriate channel as advised by mentors), to keep everyone updated and invite suggestions.

- **Regular Mentor Meetings:** I will have regular check-ins with my GSoC mentors (at least once a week, possibly more often during critical design phases). These can be via email or synchronous chats (Slack/Zoom) as preferred. In these meetings, I'll present my progress, any blockers I encountered, and plans for the next steps. I'll also actively seek mentors' guidance on Beam's best practices and any technical uncertainties (especially regarding Tecton integration).

- **Community Meetings:** If Apache Beam holds any public community sync-ups or IRC/Slack channels, I will join them. For instance, some Apache projects have bi-weekly tech talks or meetups; I will attend any that are relevant. Engaging in real-time discussions can help in quickly resolving doubts and also show the community my active participation.

- **Code Reviews and Feedback:** I will open my work as pull requests as early as reasonable (even as draft PRs). This invites Beam committers and contributors to review the code incrementally. I will promptly respond to code review comments, be open to critiques, and incorporate suggestions. This not only improves the quality of my work but also integrates me into the development workflow of the project.

- **Assist Fellow Contributors:** Community engagement is a two-way street. I plan to monitor the user mailing list or StackOverflow for any Beam questions I can answer, especially related to IO connectors or ML use-cases. Given my background, I might help with questions on Elasticsearch IO or general Beam usage if they arise. Contributing in

this way builds goodwill and sharpens my understanding of user needs.

- **Documentation Collaboration:** I will seek help from the community in reviewing the documentation I write. Often, documentation benefits from feedback by users who try to follow it. I might request a volunteer from the community to try out a connector using my docs to see if it's clear.

- **Adhering to Apache Etiquette:** I understand Apache projects value respectful, transparent communication. I will ensure all my interactions are professional and appreciative of others' time. For instance, if a community member raises a concern about design, I will discuss it objectively and consider alternatives rather than being defensive.

- **Time Management and Visibility:** To build trust, I will keep a public **project journal** (perhaps a thread on the mailing list or a shared document) where I log daily/weekly what was done. This visibility ensures the community sees steady progress and can step in with advice if I'm stuck.

- **Mid-term and Final Reports:** I will prepare a mid-term status report and a final presentation of results, possibly sharing in a community meeting or via the mailing list. This is not only for GSoC requirement but also a way to engage the community with what has been accomplished and what value it brings to Beam.

By engaging through these channels, I aim to become a familiar and trusted member of the Apache Beam community. This will not only help during GSoC but also set the stage for long-term collaboration. My previous experiences working in teams (at companies and on projects) have taught me the importance of communication — I plan to bring that same collaborative spirit to the open-source community setting.

Long-term Contribution and Future Plans

My interest in Apache Beam and the broader data processing and ML infrastructure ecosystem extends beyond the GSoC 12-week period. **I am committed to continuing as a contributor to Apache Beam** and related open-source projects in the long term. Here's how I envision my future involvement:

- **Maintaining the Connectors:** After GSoC, I will take responsibility for the maintenance of the Pinecone and Tecton connectors. This means addressing any bugs or improvements that users or reviewers identify after integration. As these connectors get used in real-world scenarios, new feature requests may emerge (e.g., support for a new Pinecone feature or an update in Tecton's API). I plan to actively follow Pinecone and Tecton's product updates and keep the Beam connectors up-to-date accordingly.

- **Feature Enhancements:** There are possibilities to extend the functionality of the connectors. For instance, for Pinecone, adding support for more advanced query options or supporting other operations like deleting vectors. For Tecton, perhaps integration with their offline store or adding caching layers. I intend to work on such enhancements post-GSoC, in collaboration with any interested community members or possibly with input from Pinecone/Tecton teams if they get involved.

- **Expanding Beam's ML I/O Ecosystem:** The proposal focuses on Pinecone and Tecton, but the idea of Beam ML connectors can be expanded. In the future, I'd be interested in adding connectors for other popular vector DBs or feature stores (for example, **Weaviate, Milvus** for vector DBs, or **Feast, Hopsworks** for feature stores). Having gained experience with this project, I could contribute to those integrations or mentor future contributors who take them on. Essentially, I'd like to help Apache Beam build a **rich ecosystem of ML data connectors**, making it a go-to platform for ML pipelines.

- **General Beam Contributions:** Beyond I/O connectors, I am keen on contributing to other parts of Beam. My backend and distributed systems expertise could be useful in improving Beam's runners or core. Perhaps I can help optimize the Python SDK, or contribute to Beam's libraries for machine learning (Beam has some ML-related transforms, I recall). I am also interested in the Beam SQL and Beam's portability framework. Post-GSoC, I'll look for areas in Beam's issue tracker that align with my skills and start picking up some of those.

- **Community Involvement:** I plan to remain active on the Beam mailing lists and forums. If the connectors I built gather interest, I might do a short presentation or a blog post about them, which could be hosted on Beam's blog if appropriate. I'm also open to mentoring new contributors; having gone through the process myself, I could guide newcomers in setting up Beam or writing their first transforms.

- **Leveraging Connectors in Real Projects:** On a personal career note, I intend to use Apache Beam and the connectors in projects I work on (either at work or personal projects). By dogfooding my contributions, I will naturally find ways to improve them. For example, I might integrate Beam+Pinecone in an AI application I prototype. This real-world usage will keep me invested and knowledgeable about the codebase.

- **Continuous Learning:** The tech world of data processing and ML moves fast. I will continue learning and possibly pursue advanced knowledge (maybe a Master's later, or online courses) in distributed data systems. As I learn, I plan to bring that knowledge into Beam. For instance, if a new efficient pattern for vector search emerges, I could help integrate it.

- **Open Source Ethos:** Ultimately, contributing to Beam is part of my larger commitment to open source. I foresee spending a portion of my time regularly contributing to open source projects I care about. Apache Beam is at the top of that list given the alignment

with my interests. Over time, I hope to become a recognized contributor, and potentially attain committer status on Apache Beam if my contributions prove valuable and consistent.

In summary, GSoC is a springboard for me into the Apache Beam community. I am enthusiastic about **continuing the journey** beyond the summer, ensuring that the project we start this summer grows and sustains. Apache Beam's mission of unifying batch and streaming and enabling portable data processing is something I strongly resonate with, and I want to be part of its growth into the ML domain. This isn't just a summer project for me – it's the beginning of a long-term collaboration.

Conclusion

In this proposal, I outlined a comprehensive plan to implement Pinecone and Tecton connectors for Apache Beam, which will significantly enhance Beam's capabilities in the machine learning domain. By enabling direct integration with a vector database and a feature store, Apache Beam can become a more powerful platform for end-to-end ML pipelines, supporting use cases from feature engineering to real-time inference with vector search. This contribution is important to the Apache Beam ecosystem because it fills a current gap – the lack of connectivity to modern ML data infrastructure – thereby expanding Beam's applicability to state-of-the-art AI applications.

I bring a strong background in backend engineering, distributed systems, and scalable architectures to this project. My experience with technologies like Elasticsearch and cloud services will be directly relevant in designing high-performance connectors that handle large-scale data. Throughout the 12-week timeline, I plan to deliver robust code, rigorous tests, and clear documentation, following Apache's high standards for quality. Just as importantly, I will engage closely with the Apache Beam community to ensure the design aligns with user needs and to establish myself as a long-term contributor.

By the end of the summer, Apache Beam will have new **Pinecone and Tecton I/O connectors** readily available. This means data scientists and engineers can, for example, use Beam to generate embeddings and store them in Pinecone in one pipeline, or pull features from Tecton for model training without leaving the Beam framework. The added convenience and capability will help Apache Beam users avoid writing glue code and instead focus on the core logic of their ML workflows. In essence, this project will **empower Apache Beam to serve as a unified pipeline solution for advanced ML use-cases**, strengthening its position in the ecosystem of data tools.

Thank you for considering my proposal. I am excited about the opportunity to contribute to Apache Beam through GSoC 2025. I am confident that, with my skills and dedication, I can successfully implement the ML Vector DB/Feature Store integrations for Pinecone and Tecton. I look forward to the possibility of making this contribution and continuing to work with the Apache

Beam community well beyond the summer. Together, we will help Apache Beam bridge the gap between data processing and machine learning, driving the project forward into new domains.