Additional Considerations for Application

---

7. Enhanced Metadata

To improve reporting and downstream processing, the following metadata fields can be added:

Additional Metadata Fields:

- **Protein Source:** Whether the sequence originates from a database (e.g., PDB, UniProt) or user-uploaded.
- **File Validation Status:** Checksum hash (e.g., SHA256) to verify file integrity.
- **Protein Family & Classification:** Classification based on known databases (e.g., CATH, SCOP).
- **Post-Processing Metrics:** Error rates, sequence confidence scores.
- **Computation Resource Metrics:** CPU/GPU utilization, memory usage.
- **Processing Logs & Timestamps:** Record each step's duration for debugging and profiling.

Implementation Considerations:

- Store metadata in **JSON format** (e.g., `outputs/1bey_metadata.json`).
- Use a **database** (e.g., PostgreSQL, MongoDB) for structured storage.
- Ensure metadata is **easily retrievable** for future processing.

---

8. Pipeline Trigger via UI

To allow users to trigger the pipeline from a graphical interface, the following design elements could be considered:

UI Workflow:

1. **File Upload Module**: Users upload `.pdb` files via a web-based UI.
2. **Processing Status Indicator**: Real-time progress updates via WebSockets.
3. **Results Display**: Metadata and visualization of protein chains.
4. **Download Option**: Button to download processed metadata files.
5. **Error Handling UI**: Display validation errors if the file is incorrect.

Technology Options:

- **Frontend:** React, Angular, or a simple Flask/Jinja2-based UI.
- **Backend Communication:** REST API (`/process-pdb/`) or WebSockets.
- **Task Management:** Use **Celery** with Redis for background processing.

---

9. Performance Optimization

1. Deploying in Cloud (AWS, GCP, Azure)

- **Serverless Approach:** AWS Lambda for lightweight inference.
- **Compute-Optimized EC2 Instances:** For heavy protein processing.
- **S3 Storage:** Store large `.pdb` and metadata files efficiently.

2. Using Kubernetes for Scalability

- **Microservices-based approach**:
    - Deploy **FastAPI** app in **Kubernetes Pods**.
    - Use **Horizontal Pod Autoscaler (HPA)** to dynamically scale based on CPU/GPU load.
    - Manage job execution using **Kubernetes Jobs**.
- **Load Balancer:**
    - Use **NGINX Ingress Controller** to handle API requests.
    - Deploy **Redis Queue** to manage incoming processing tasks.

3. Optimizing Sequence Extraction & Inference

- **Parallel Processing:** Use **Dask** or **Ray** to process multiple PDBs concurrently.
- **GPU Acceleration:** If using deep learning, optimize **Torch/TensorFlow** with GPU (CUDA).
- **Caching Mechanism:** Store processed sequences in **Redis** or **SQLite** for faster retrieval.

---

10. Storage Solutions for Scalability

Scenario A: Light & Heavy Chain Files Used Concurrently

- **Solution:** Implement a **shared file system** (e.g., **AWS EFS** or **Google Cloud Filestore**) to allow multiple processes to access `.pdb` files simultaneously.
- **Alternative:** Store extracted sequences in **MongoDB** with a reference to the original PDB file.

Scenario B: Large-Scale Inference (Thousands of Proteins)

- **Solution:** Use a **data lake architecture**:
    - Store raw PDB files in **AWS S3** or **Google Cloud Storage**.
    - Use **Apache Spark** to process and analyze large datasets in parallel.
    - Implement **database partitioning** to manage metadata efficiently.
