Universidad *[University Name]*

Department of *[Department Name]*

PROTEA: A Framework for Scalable Protein Functional Annotation

Master's Thesis

Author
Francisco Percan

Supervisor
[Supervisor Name]

Co-Supervisor
[Co-Supervisor Name]

Academic Year 2025–2026

Abstract

Predicting the biological function of proteins at scale remains one of the central challenges in computational biology. Experimental characterisation cannot keep pace with the rate at which new sequences are deposited in public databases, leaving the vast majority of known proteins with no experimentally validated functional annotation.

This thesis presents **PROTEA** (*PROtein funcTional annotation framEwork and plAtform*), a distributed system that automates the full pipeline from raw sequence to structured Gene Ontology (GO) term predictions. PROTEA integrates large language model embeddings, approximate nearest-neighbour search, and evidence-transfer from Universal Protein Resource (UniProt) reviewed annotations to produce ranked, interpretable functional predictions. The system is designed for reproducibility, horizontal scalability, and ease of use by researchers without a background in machine learning infrastructure.

Experimental evaluation on a held-out subset of the Swiss Protein database (reviewed section of UniProt) (Swiss-Prot) database demonstrates that PROTEA achieves competitive performance relative to existing annotation transfer methods, while providing additional feature-engineering signals — pairwise sequence alignment statistics and taxonomic distance — that improve prediction confidence.

Keywords: protein function prediction, gene ontology, sequence embeddings, approximate nearest-neighbour search, bioinformatics, distributed systems.

Resumen

[Spanish abstract — to be written]

Acknowledgements

[To be written.]

Contents

List of Figures

List of Tables

Acronyms

BP	Biological Process. 5
CC	Cellular Component. 5
DAG	Directed Acyclic Graph. 5
GAF	GO Annotation File. 26
GO	Gene Ontology. iii, 1, 2, 5
HMM	Hidden Markov Model. 1, 10
kNN	k-Nearest Neighbours. 1
LCA	Lowest Common Ancestor. 6, 27
MF	Molecular Function. 5
NW	Needleman–Wunsch. 6, 20, 26
OBO	Open Biological and Biomedical Ontologies. 5
PDB	Protein Data Bank. 4
pLM	Protein Language Model. 1, 6, 11, 12
SW	Smith–Waterman. 6, 20, 26
Swiss-Prot	Swiss Protein database (reviewed section of UniProt). iii, 4
TrEMBL	Translated EMBL Nucleotide Sequence Data Library. 4
UniProt	Universal Protein Resource. iii, 4

Chapter 1

Introduction

1.1 Motivation

The pace at which biological sequence data accumulates has long outstripped our capacity to characterise proteins experimentally. As of early 2026, UniProt contains more than 250 million sequences, yet fewer than 0.02 % carry experimentally validated functional annotations [24]. The gap is not merely quantitative: the proteins that remain unannotated are disproportionately found in under-studied organisms, precisely where novel biology — and potential therapeutic targets — are most likely to reside.

Computational function prediction attempts to bridge this gap by transferring knowledge from characterised proteins to unknown ones. Classical approaches based on sequence similarity (BLAST, profile Hidden Markov Models (HMMs)) remain widely used, but their accuracy degrades for distantly related sequences. More recently, Protein Language Models (pLMs) trained on hundreds of millions of sequences have produced dense vector representations that capture structural and functional signals well beyond what classical alignment can detect.

PROTEA was designed to operationalise these advances: to take a set of protein sequences as input and produce, at scale, ranked GO term predictions with supporting evidence, while remaining accessible to domain experts who are not machine-learning engineers.

1.2 Research Questions

This thesis addresses the following questions:

RQ1. Can embedding-based k-Nearest Neighbours (kNN) annotation transfer match or exceed alignment-based methods for GO term prediction, particularly at low sequence identity?

RQ2. Do pairwise alignment statistics and taxonomic distance provide complementary signals that improve the ranking of candidate annotations?

RQ3. How can a reproducible, scalable annotation pipeline be designed so that researchers can re-run analyses as ontologies and databases are updated?

1.3 Contributions

The main contributions of this work are:

- **PROTEA**: an open-source distributed platform implementing the full annotation pipeline, from FASTA upload to structured GO predictions with optional feature engineering.
- A **versioned data model** for ontology snapshots and annotation sets that enables reproducible retrospective analyses.
- A **feature-engineering module** that augments embedding similarity with Needleman–Wunsch / Smith–Waterman alignment scores and NCBI-taxonomy-based distances.
- An **empirical evaluation** comparing PROTEA predictions against baseline methods on a held-out Swiss-Prot subset.

1.4 Thesis Structure

The remainder of this document is organised as follows. Chapter 2 introduces the biological concepts underpinning the work. Chapter 3 surveys existing computational approaches to protein function prediction. Chapter 4 presents the architecture of PROTEA. Chapter 5 describes key implementation decisions. Chapter 6 reports the experimental results. Chapter 7 summarises the findings and outlines future directions.

Chapter 2

Biological Background

This chapter provides the biological foundation required to understand the problem that PROTEA addresses. Readers with a background in molecular biology may skip directly to chapter 3.

2.1 Proteins: Structure and Function

Proteins are macromolecules composed of chains of amino acids linked by peptide bonds. The linear sequence of amino acids — encoded by the genome — is referred to as the *primary structure*. This sequence folds, driven by thermodynamic forces, into a specific three-dimensional conformation that determines the protein's biological activity.

Protein functions are extraordinarily diverse: enzymes catalyse chemical reactions, structural proteins provide mechanical support, signalling proteins relay information between cells, and transport proteins shuttle molecules across membranes. Understanding *what* a protein does — its molecular function, the biological process it participates in, and the cellular compartment where it operates — is fundamental to biology and medicine.

2.1.1 Amino Acids and the Sequence Space

Twenty standard amino acids, differing in their side-chain chemistry, constitute the alphabet of protein sequences. A protein of n residues can in principle take 20^n distinct sequences, an astronomically large space. Yet the sequences that fold into stable, functional structures represent a tiny, structured subset of this space, one in which similar sequences tend to adopt similar folds and perform similar functions.

2.1.2 Sequence Identity and Homology

Two proteins are said to be *homologous* if they share a common evolutionary ancestor. Homology is typically inferred from sequence similarity: proteins with high pairwise sequence identity are almost always homologous and usually perform the same function. As sequence identity drops below roughly 30 %, however, the relationship between similarity and function becomes ambiguous — proteins may be structurally similar yet functionally diverged, or functionally convergent with unrelated folds.

2.2 Protein Databases

2.2.1 UniProt

UniProt is the central repository for protein sequence and functional information [24]. It is divided into two sections:

Swiss-Prot A manually curated, high-quality database. Each entry has been reviewed by expert annotators and contains experimental or computationally inferred functional annotations, with explicit evidence codes.

Translated EMBL Nucleotide Sequence Data Library (TrEMBL) An automatically annotated, computationally analysed database derived from genome translations. It is several orders of magnitude larger than Swiss-Prot but of lower average annotation quality.

Each protein in UniProt is identified by a stable accession number. Canonical isoforms are represented by a bare accession (e.g. `P12345`); alternative splicing variants are denoted `P12345-2`, `P12345-3`, and so on.

2.2.2 Protein Data Bank

The Protein Data Bank (PDB) archives three-dimensional structures of biological macromolecules determined experimentally by X-ray crystallography, cryo-electron microscopy, or NMR [3]. Although PROTEA operates primarily on sequences, PDB structures are an important source of ground-truth functional information and are used indirectly through UniProt cross-references.

2.3 The Gene Ontology

2.3.1 Overview

The GO is a standardised, species-independent controlled vocabulary for describing gene product attributes [2, 11]. It is structured as three independent Directed Acyclic Graphs (DAGs), one for each of the three top-level *aspects*:

Molecular Function (MF) (GO:0003674) The biochemical activity of the gene product, independent of where or when it occurs (e.g. *ATP binding*, *serine-type endopeptidase activity*).

Biological Process (BP) (GO:0008150) The larger biological programme to which the activity contributes (e.g. *apoptotic process*, *DNA repair*).

Cellular Component (CC) (GO:0005575) The place in the cell where the gene product is active (e.g. *nucleus*, *plasma membrane*).

Terms within each aspect are connected by *is-a* and *part-of* relationships, forming a hierarchy from broad root terms to highly specific leaves. A protein annotated with a term is implicitly annotated with all of its ancestors up to the root — the *true path rule*.

2.3.2 Evidence Codes

Every GO annotation carries an evidence code that indicates the type of support for the association. Codes range from experimental (e.g. `EXP`, `IDA`, `IMP`) to computationally inferred (`ISS`, `IEA`) and author-stated (`TAS`, `NAS`). Experimental evidence codes are the gold standard; `IEA` (Inferred from Electronic Annotation) is the most common but least reliable.

2.3.3 Ontology Versioning

The GO is actively maintained and released regularly in Open Biological and Biomedical Ontologies (OBO) format. New terms are added, obsolete terms are deprecated, and relationships are refined with each release. Reproducible research therefore requires that analyses record which ontology version was used. PROTEA stores each imported release as an `OntologySnapshot` and associates all annotation sets and predictions with a specific snapshot.

2.4 Sequence Alignment

Pairwise sequence alignment is the classical approach to measuring sequence similarity. Two algorithms are central to this thesis:

Needleman–Wunsch (Needleman–Wunsch (NW)) A global alignment algorithm that aligns two sequences end-to-end, maximising a cumulative substitution score minus gap penalties [18]. It is suitable for comparing proteins of similar length.

Smith–Waterman (Smith–Waterman (SW)) A local alignment algorithm that finds the highest-scoring contiguous aligned region, ignoring the tails of each sequence [22]. It is more appropriate when one sequence is a domain or fragment of the other.

Both algorithms use a *substitution matrix* — typically BLOSUM62 for protein sequences [12] — to score residue substitutions based on their observed frequency in alignments of related proteins.

2.5 Protein Language Models and Embeddings

PLMs are neural networks trained on large corpora of protein sequences using self-supervised objectives borrowed from natural language processing. Models such as ESM-2 [16] and ProtTrans [9] learn to predict masked residues, in doing so developing internal representations that capture evolutionary, structural, and functional information.

A protein sequence can be passed through such a model to obtain a dense vector (embedding) that encodes its properties in a continuous latent space. Proteins with similar functions tend to cluster together in this space even when their sequences have diverged below the homology detection threshold of alignment-based tools. This property makes embeddings attractive for annotation transfer.

2.6 Taxonomic Classification

Biological organisms are classified into a hierarchy — the NCBI taxonomy — that reflects their evolutionary relationships [21]. Each node is assigned a numeric *taxon ID*. The distance between two organisms can be defined operationally as the number of edges between them through their Lowest Common Ancestor (LCA) in the taxonomy tree.

Taxonomic distance is a useful co-variate for function prediction: a protein from a closely related organism is more likely to perform the same function than one from a distant lineage, even when embedding similarity is comparable.

Chapter 3

Related Work

Automated protein function prediction has been an active research area for more than two decades. Methods can be grouped into four broad families: sequence alignment transfer, profile and domain matching, supervised machine learning, and embedding-based approaches. This chapter surveys each family, identifies their limitations, and contextualises PROTEA within the landscape.

3.1 Alignment-Based Annotation Transfer

The foundational insight of alignment-based methods is that sequence similarity implies functional similarity. If a query protein shares high sequence identity with a characterised protein, one can transfer the known annotations to the query.

BLAST [1] remains the most widely deployed tool for this purpose. It uses a heuristic seed-and-extend strategy to find local alignments in sublinear time. `BLASTP` (protein–protein) identifies homologues in Swiss-Prot and transfers their GO terms to the query. This simple approach is accurate when identity exceeds roughly 50 % and degrades progressively as identity falls. Below the "twilight zone" of ≈30 % identity, alignment scores lose statistical power and functional inference becomes unreliable [20].

DIAMOND [5] offers BLAST-comparable sensitivity at orders-of-magnitude higher throughput, making it practical for genome-scale annotation. Its speed advantage comes from reduced-alphabet seeds and better SIMD utilisation; the fundamental twilight-zone limitation remains.

Annotation propagation rules add another layer of nuance: not all GO terms are equally transferable. Terms annotated with `IEA` codes can be propagated freely, while experimental annotations require explicit curation policies. The CAFA community has established benchmark protocols that penalise propagation errors [26].

3.2 Profile and Domain Methods

To extend sensitivity below the twilight zone, profile methods compare a query against position-specific models built from multiple sequence alignments of protein families.

HMMER [8] implements profile HMMs for this purpose. A family is represented as a probabilistic model over residue frequencies at each position, capturing conserved and variable sites more faithfully than a single representative sequence. HMMER achieves greater sensitivity than BLAST at the cost of requiring pre-built profiles.

InterPro [4] integrates over a dozen family and domain databases — Pfam, PRINTS, ProSite, TIGRFAM, and others — into a unified annotation resource. A protein is assigned InterPro entries whenever it matches any member database; GO terms are then inferred via curated mappings. InterPro annotation is one of the most reliable computational methods available, but it is limited to protein families and domains that have been explicitly curated and modelled.

Cascade homology search [19] iteratively extends sensitivity by feeding BLAST hits into further BLAST searches (PSI-BLAST) or HMM construction, at the cost of introducing false positives in remote homology regimes.

3.3 Supervised Machine Learning Approaches

Supervised methods learn classifiers — one per GO term of interest — from labelled protein–function pairs. Early approaches used HMMs or support vector machines on curated feature sets (amino acid composition, physicochemical properties, secondary structure predictions). More recent work uses deep neural networks applied directly to sequences.

DeepGO [15] and its successor **DeepGOPlus** [14] train convolutional networks on sequence k-mers and combine the output with sequence similarity scores. They demonstrate that learned sequence features and alignment evidence are complementary: the two signals fuse into a more reliable predictor than either alone.

NetGO [25] augments sequence-based models with protein–protein interaction network features, achieving state-of-the-art results in the CAFA3 challenge. Its dependency on interaction data limits applicability for poorly studied organisms.

A fundamental challenge for supervised methods is the extreme imbalance and hierarchy of the GO label space. The number of proteins annotated with any specific GO term decreases rapidly as one descends the ontology DAG; for many leaf terms, fewer than ten training examples exist. Hierarchical loss functions and ontology-aware evaluation metrics (Fmax, AUPR) have been developed to address this [7].

3.4 Protein Language Model Embeddings

The advent of large-scale self-supervised pLMs has opened a qualitatively different avenue for function prediction. Models trained on hundreds of millions of sequences develop internal representations — embeddings — that capture evolutionary relationships, secondary and tertiary structure propensities, and functional sites, without any explicit structural or functional supervision.

ESM-1b and **ESM-2** [16] (Meta AI) are transformer models trained on UniRef50/90. ESM-2 scales from 8M to 15B parameters; the larger variants generate embeddings that outperform BLAST-based transfer at low-identity regimes. ESM-2 embeddings serve as the primary representation in PROTEA.

ProtTrans [9] provides a suite of encoder models (ProtBERT, ProtXLNet, ProtT5) trained on BFD and UniRef100 using distributed HPC resources. ProtT5-XL achieves the best per-residue and per-protein accuracy among the ProtTrans variants.

Annotation transfer via nearest neighbours is the inference strategy most naturally paired with embeddings. Given a set of reference proteins with known annotations and their embeddings, the k nearest reference neighbours of a query (by cosine or Euclidean distance) are retrieved and their annotations aggregated to form a prediction. This approach was explored in Littmann et al. [17] and forms the core inference mechanism of PROTEA.

Scalability is the central engineering challenge for embedding-based transfer. Swiss-Prot alone contains over 570,000 entries; computing exact cosine distances against all of them for a large query set requires $O(NQ)$ dot products, which becomes infeasible at scale. Approximate nearest-neighbour libraries such as FAISS [13] reduce this to $O(Q\sqrt{N})$ with minimal accuracy loss for IVFFlat indices.

3.5 CAFA Benchmark and Evaluation Standards

The Critical Assessment of Functional Annotation (CAFA) [26] challenge provides the community standard for evaluating function prediction methods. Participants submit predictions for a set of target proteins; ground truth is established retrospectively from experimental annotations deposited after the submission deadline.

The primary metric is $F_{\max}$: the maximum F_1 score across all prediction confidence thresholds, computed separately for each GO aspect. AUPR (area under the precision–recall curve) and Smin (semantic distance) are complementary measures. PROTEA's evaluation in chapter 6 follows CAFA conventions.

3.6 Annotation Platforms and Pipelines

Several platforms have been developed to operationalise function prediction at genome scale.

Argot2 [10] combines BLAST and HMMER hits with a graph propagation scheme over the GO DAG. It produces confidence scores by integrating multiple lines of evidence.

PANNZER2 [23] is a web server that combines BLAST-based nearest-neighbour transfer with a SANS scoring model. It provides an interactive interface and supports batch submission.

eggNOG-mapper [6] maps sequences against clusters of orthologous groups (COGs) and transfers functional annotations from the eggNOG database. It is widely used in comparative genomics pipelines.

None of these platforms provides a reproducible, versioned annotation pipeline backed by a relational data model that records which ontology version, reference annotation set, and embedding configuration produced each prediction — a gap that PROTEA addresses directly.

3.7 Summary and Positioning of PROTEA

Table 3.1 summarises the methods discussed above along four axes relevant to this thesis.

Table 3.1: Comparison of protein function prediction approaches.

Method	Low-identity	Scalable	Versioned	Open source
BLAST transfer	×	✓	×	✓
HMMER/InterPro	✓	✓	×	✓
DeepGO/DeepGOPlus	✓	✓	×	✓
PANNZER2	×	✓	×	✓
ESM-2 kNN transfer	✓	∼	×	✓
PROTEA	✓	✓	✓	✓

PROTEA occupies a niche that none of the surveyed systems fills: an end-to-end distributed pipeline that combines pLM embeddings with optional alignment and taxonomy features, records all versioning information in a relational schema, and exposes the full pipeline through a REST API and web interface.

Chapter 4

System Design

This chapter describes the architecture of PROTEA at the level of design decisions. Implementation details — specific libraries, migration tooling, frontend components — are deferred to chapter 5.

4.1 Requirements and Design Goals

The design of PROTEA is governed by five requirements derived from the limitations identified in chapter 3:

R1 — Reproducibility. A prediction produced today must be exactly reproducible in the future. This requires recording the ontology version, reference annotation set, and embedding model configuration used for every prediction run.

R2 — Scalability. The system must handle reference sets of hundreds of thousands of proteins and query sets of thousands without holding all data in memory simultaneously.

R3 — Separation of concerns. Domain logic (what to compute), execution flow (how jobs are dispatched and tracked), and infrastructure (database, message queue) must be independently replaceable.

R4 — Observability. Every job must produce a structured audit trail so that failures can be diagnosed without replaying the computation.

R5 — Accessibility. Researchers without machine-learning infrastructure expertise must be able to submit sequences and retrieve predictions through a web interface or a REST API.

4.2 High-Level Architecture

PROTEA decomposes into four horizontal layers, illustrated in fig. 4.1:

Presentation layer. A Next.js single-page application exposes the user-facing workflow: uploading FASTA files, triggering pipeline jobs, monitoring progress, and downloading results.

API layer. A FastAPI application provides a REST interface. All state mutations flow through this layer; it writes job requests to PostgreSQL and publishes messages to RabbitMQ.

Worker layer. A set of Python worker processes consume messages from RabbitMQ queues. Workers are stateless with respect to domain logic — they resolve and execute registered operations and update job state in the database.

Data layer. PostgreSQL (extended with the pgvector extension for vector storage) is the single source of truth. RabbitMQ is used exclusively for task dispatch; it carries no persistent state.

[Architecture diagram — to be added]

Figure 4.1: High-level architecture of PROTEA.

4.3 The Operation Abstraction

The central design principle of PROTEA is the *Operation protocol*: every unit of domain logic is a class that implements exactly two members:

- `name: str` — a unique string identifier used to route messages.
- `execute(session, payload, *, emit) -> OperationResult` — the computation itself.

Operations receive a SQLAlchemy session and a validated Pydantic payload; they report progress through an `emit` callback that writes `JobEvent` rows to the database. Operations do not manage their own sessions, do not publish messages, and do not know about the queue infrastructure. This strict interface makes them independently testable and trivially substitutable.

The `OperationRegistry` maps string names to instances at startup. Workers resolve the correct operation by name when they receive a message, enabling new operations to be added without modifying any worker code.

4.4 Job Lifecycle and Worker Patterns

PROTEA distinguishes two worker patterns that correspond to different observability requirements:

4.4.1 QueueConsumer (tracked jobs)

Long-running, user-visible jobs — inserting proteins, loading ontologies, computing embeddings, predicting GO terms — are submitted through the API as `Job` rows with a JSONB payload. The workflow is:

1. The API creates a `Job` row in state `QUEUED` and publishes the job UUID to the appropriate RabbitMQ queue.
2. A `QueueConsumer` worker receives the UUID, opens a *claim session*, transitions the job to `RUNNING`, and flushes the `started_at` timestamp. The claim session is committed and closed before execution begins.
3. A separate *execute session* resolves the operation and calls `execute()`. On success the job transitions to `SUCCEEDED`; on exception to `FAILED`, with the error code and message stored on the row.

The two-session design ensures that the claim is durable even if the execute session rolls back. `JobEvent` rows, written via `emit()`, form an append-only audit log that records progress milestones, error context, and structured diagnostic fields.

4.4.2 OperationConsumer (fire-and-forget sub-tasks)

GPU-intensive batch tasks — running the embedding model on a group of sequences, or computing kNN predictions for a batch — do not create child `Job` rows. Instead, the coordinator operation publishes *operation messages* (containing the payload directly, not a UUID) to dedicated queues. `OperationConsumer` workers execute these messages and report progress via an atomic increment on the parent job's counter, avoiding write contention from hundreds of concurrent batches.

Figure 4.2 illustrates the two patterns and the queue topology.

4.4.3 Parent-Child Job Hierarchy

Coordinator operations (`compute_embeddings`, `predict_go_terms`) split work across many parallel workers using a parent-child pattern. The coordinator returns `OperationResult(deferred=True)`, which signals `BaseWorker` to *not* transition the parent job to `SUCCEEDED` immediately. Instead, the parent remains in state `RUNNING` while batch workers process their messages independently.

The `Job` model carries two progress fields for this purpose: `progress_total` (set by the coordinator to the number of batches dispatched) and `progress_current` (incremented atomically by each write worker). When the last write worker detects `progress_current` = `progress_total`, it closes the parent job as `SUCCEEDED`. This atomic counter design avoids write contention from hundreds of concurrent batches updating the same row.

4.4.4 RetryLaterError

When a shared resource is unavailable — for instance, the GPU is already occupied by a running embedding job — an operation raises `RetryLaterError` instead of failing permanently. `BaseWorker` catches this exception and:

1. Resets the job status back to `QUEUED`.
2. Writes a `job.retry_later` event recording the reason and the requested delay.
3. Re-publishes the job UUID to its queue after the specified delay (typically 60 seconds).

The embedding coordinator queue (`protea.embeddings`) is serialised precisely because of this mechanism: at most one embedding coordinator runs at a time, with subsequent requests queued and retried automatically.

4.4.5 Cancellation

`POST /jobs/{id}/cancel` transitions a `QUEUED` or `RUNNING` job to `CANCELLED`. If the job is already in a terminal state (`SUCCEEDED`, `FAILED`, or `CANCELLED`) the request is a no-op. Any queued child jobs are also cancelled atomically in the same transaction.

Cancellation of a `RUNNING` job is a *soft cancel*: the database row is marked `CANCELLED` immediately, but the worker process is not interrupted. The worker will still complete its operation and attempt a final status write; however, because `CANCELLED` is already committed, the frontend reflects the cancelled state regardless of how the worker exits.

[Job lifecycle / queue topology diagram — to be added]

Figure 4.2: Job lifecycle and queue routing in PROTEA.

4.5 Queue Topology

PROTEA routes messages across seven RabbitMQ queues, each with a distinct semantic purpose:

`protea.ping` Smoke-test queue. Used to verify end-to-end connectivity.

`protea.jobs` General-purpose job queue. Handles protein insertion, metadata fetch, ontology loading, annotation loading, and the coordinator phase of embeddings and predictions.

`protea.embeddings` Serialised embedding coordinator queue. At most one coordinator runs at a time; if the GPU is busy the coordinator raises `RetryLaterError` and is re-queued with a 60-second delay.

`protea.embeddings.batch` GPU inference queue. Each message carries a batch of sequences; `OperationConsumer` workers run the forward pass of the protein language model and publish results to the write queue.

`protea.embeddings.write` Bulk insert queue. Workers receive computed embedding vectors and write them to the `SequenceEmbedding` table via pgvector.

`protea.predictions.batch` kNN prediction queue. Each message carries a batch of query embeddings; workers search the reference index and transfer GO annotations.

`protea.predictions.write` Bulk prediction insert queue. Workers write `GOPrediction` rows for a batch of query–reference pairs.

This topology separates GPU-bound inference, I/O-bound database writes, and CPU-bound coordination into independent queues that can be scaled independently (e.g. `manage.sh scale protea.predictions.batch 4`).

4.6 Data Model

The relational schema is organised into five logical groups:

4.6.1 Sequence and Protein

`Sequence` stores deduplicated amino acid strings, keyed by MD5 hash. `Protein` stores one row per UniProt accession; multiple accessions (canonical and isoforms) may reference the same `Sequence`. This deduplication prevents redundant embedding computation.

4.6.2 Ontology and Annotations

`OntologySnapshot` records one complete GO release (OBO format), versioned by the `obo_version` string found in the OBO file header. `GOTerm` and `GOTermRelationship` store the DAG within a snapshot.

`AnnotationSet` groups a batch of `ProteinGOAnnotation` rows by source (`goa` or `quickgo`) and links them to an `OntologySnapshot`. This design enables side-by-side comparison of annotation sets from different sources or dates.

4.6.3 Embeddings

`EmbeddingConfig` is an immutable recipe: it records the model identifier, chunking strategy, pooling method, and any other parameters that affect the embedding geometry. Its UUID primary key is stable; changing any parameter creates a new configuration.

`SequenceEmbedding` stores one pgvector `VECTOR` per (`sequence`, `config`, `chunk`) triple. Chunking support allows sequences that exceed the model's context window to be split into overlapping segments.

4.6.4 Predictions

`PredictionSet` is the result container, linking a query set, an embedding configuration, an annotation set, and an ontology snapshot. `GOPrediction` stores one row per (query protein, GO term) pair, including the cosine distance from the nearest reference neighbour and the optional feature-engineering columns described in section 5.5.

4.6.5 Query Sets

`QuerySet` represents a user-uploaded FASTA dataset submitted for custom prediction queries. Each `QuerySetEntry` row preserves the original accession header from the FASTA file and links to the deduplicated `Sequence` row, reusing existing sequences when the amino-acid string is already present in the database. Query sets are created via `POST /query-sets` and can be referenced in subsequent embedding computation and prediction jobs.

4.6.6 Jobs

`Job` implements a state machine with states `QUEUED`, `RUNNING`, `SUCCEEDED`, `FAILED`, and `CANCELLED`. `JobEvent` is an append-only log of timestamped, structured events emitted during execution. Both tables use JSONB columns for extensible payloads and metadata.

4.7 Embedding Pipeline

The embedding pipeline runs in three phases coordinated through the queue topology described above:

1. **Coordinator** (`compute_embeddings`): queries the database for all sequences in the target set that do not yet have an embedding for the requested `EmbeddingConfig`; partitions them into fixed-size batches; publishes one operation message per batch to `protea.embeddings.batch`.

2. **Batch inference** (`compute_embeddings_batch`): loads the protein language model on the configured device; tokenises the sequence batch; runs a forward pass; mean-pools the per-residue representations into a single vector per chunk; publishes the computed vectors to `protea.embeddings.write`.

3. **Store** (`store_embeddings`): receives the computed vectors and performs a bulk upsert into `SequenceEmbedding`.

The coordinator is serialised (one at a time per the `protea.embeddings` queue) to prevent concurrent model loads from exhausting GPU memory. Batch and write workers can be scaled horizontally.

4.8 Prediction Pipeline

The prediction pipeline mirrors the structure of the embedding pipeline:

1. **Coordinator** (`predict_go_terms`): loads the reference embeddings (all proteins in the annotation set that have embeddings) into a process-level cache compressed to float16; partitions the query set into batches; publishes one operation message per batch to `protea.predictions.batch`.
2. **Batch prediction** (`predict_go_terms_batch`): for each query, retrieves the k nearest reference proteins using the configured search backend (NumPy exact cosine search or FAISS approximate search); transfers GO annotations from each neighbour; optionally computes pairwise alignment statistics (NW and SW via the parasail library) and taxonomic distance (via ete3 and the NCBI taxonomy); publishes the results to `protea.predictions.write`.
3. **Store** (`store_predictions`): bulk-inserts `GOPrediction` rows.

The reference cache is held at the process level and is keyed by (embedding config, annotation set). It holds at most one entry to bound memory usage; a restart reloads from the database automatically.

4.9 REST API Design

The API follows REST conventions. Resources are grouped into six routers:

`/proteins` Insert proteins from UniProt accession lists or FASTA.

`/annotations` Load ontology snapshots and annotation sets.

`/embeddings` Manage embedding configurations, trigger computation, submit prediction jobs, and export results.

`/query-sets` Upload FASTA files and manage user query datasets.

`/jobs` Track job state and stream structured events.

`/admin` Administrative maintenance tasks.

Session injection follows FastAPI's dependency system: `app.state.session_factory` is set at startup and injected into each request handler, keeping router code free of infrastructure imports.

Chapter 5

Implementation

This chapter describes the concrete technology choices and key implementation details behind the design presented in chapter 4.

5.1 Project Structure

The PROTEA codebase is organised into three top-level packages that mirror the architectural layers described in chapter 4:

`protea/api/` FastAPI application and routers (`jobs`, `proteins`, `annotations`, `embeddings`, `query_sets`, `admin`).

`protea/core/` Pure domain logic: the `Operation` protocol and `OperationRegistry` contracts, all nine registered operations, the kNN search backends, feature engineering utilities, and shared HTTP helpers.

`protea/infrastructure/` SQLAlchemy ORM models, RabbitMQ consumer and publisher, the session context manager, and the configuration loader.

Two additional directories complete the repository:

`apps/web/` The Next.js frontend application.

`scripts/` Operational tooling: `manage.sh` (process manager), `worker.py` (worker entry point that registers all operations), `init_db.py` (schema initialisation), and `run_one_job.py` (manual job runner for debugging).

This layout enforces the dependency direction: `api` and `workers` depend on `core` and `infrastructure`; `core` has no dependency on either of the other two.

5.2 Technology Stack

Table 5.1 summarises the main components of the PROTEA stack.

Table 5.1: PROTEA technology stack.

Component	Technology	Version
API framework	FastAPI	0.115+
ORM / migrations	SQLAlchemy 2.0 + Alembic	2.0 / 1.13
Database	PostgreSQL 16 + pgvector	16 / 0.7
Message broker	RabbitMQ + aio-pika	3.x / 9.x
Data validation	Pydantic v2	2.x
Protein LM inference	Hugging Face Transformers	4.x
Alignment	parasail-python (BLOSUM62)	1.x
Taxonomy	ete3 + NCBITaxa	3.x
ANN search	NumPy / FAISS	—
Frontend	Next.js 19 + Tailwind v4	19 / 4
Dependency mgmt.	Poetry	1.x

All Python dependencies are declared in `pyproject.toml` with pinned version ranges; `poetry.lock` guarantees reproducible installs across environments. The `dev` dependency group adds pytest, pytest-cov, and related tooling without polluting the production image.

5.3 Database Schema and Migrations

The database schema is managed with Alembic. Migration scripts are generated from SQLAlchemy ORM metadata via `alembic revision -autogenerate` and stored under `alembic/versions/`. The Alembic `env.py` reads the database URL from `protea/config/system.yaml` (or the `PROTEA_DB_URL` environment variable), ensuring consistency with the application configuration.

The pgvector extension is enabled at database initialisation time:

Listing 5.1: Enabling pgvector.

```sql
CREATE EXTENSION IF NOT EXISTS vector;
```

`SequenceEmbedding` columns are declared as `VECTOR(n)` where n is determined by the embedding model (e.g. 1280 for ESM-2 650M, 2560 for ESM-2 3B). Although pgvector supports index types for nearest-neighbour queries (HNSW, IVFFlat), PROTEA intentionally does not use them for kNN search — querying through pgvector at the scale of hundreds of thousands of vectors introduces unacceptable latency. Instead, reference embeddings are loaded into Python (NumPy or FAISS) at prediction time.

5.3.1 Session Management

All database access goes through `session_scope()`, a context manager that commits on normal exit and rolls back on exception:

Listing 5.2: Session context manager.

```
@contextmanager
def session_scope(factory):
    session = factory()
    try:
        yield session
        session.commit()
    except Exception:
        session.rollback()
        raise
    finally:
        session.close()
```

The two-session pattern in `BaseWorker` (claim session, execute session) relies on this primitive: the claim commits independently of the execute, so a crash during execution never rolls back the `RUNNING` state transition.

5.4 Operations

PROTEA ships eight operations registered at worker startup:

`ping` Smoke test. Returns immediately with a success result.

`insert_proteins` Paginates the UniProt REST API using cursor-based FASTA streaming. Sequences are deduplicated by MD5 hash before upsert; proteins are upserted by accession. Exponential backoff with jitter and `Retry-After` header handling are implemented in the shared `UniProtHttpMixin`.

`fetch_uniprot_metadata` Downloads TSV functional annotation data from UniProt and upserts `ProteinUniProtMetadata` rows keyed by canonical accession.

`load_ontology_snapshot` Downloads a GO OBO file and populates `OntologySnapshot`, `GOTerm`, and `GOTermRelationship` rows. The `obo_version` field carries a unique constraint so that re-importing the same release is idempotent.

`load_goa_annotations` Bulk-loads a GO Annotation File (GAF) file. Annotations are filtered against canonical accessions present in the database, avoiding orphaned foreign keys.

`load_quickgo_annotations` Streams GO annotations from the QuickGO bulk download API (paginated TSV). Supports optional ECO→GO evidence code mapping and per-page commits to bound transaction size.

`compute_embeddings` Coordinator operation described in section 4.7.

`predict_go_terms` Coordinator operation described in section 4.8.

5.5 Feature Engineering

When a prediction job is submitted with `compute_alignments=true` or `compute_taxonomy=true`, the batch prediction operation augments each query–reference pair with additional numerical features.

5.5.1 Pairwise Alignment

For each (query, reference) pair, PROTEA computes global (NW) and local (SW) alignments using the `parasail` library with BLOSUM62 substitution scores and gap-open/gap-extend penalties of $-11/-1$. The following derived statistics are stored as columns on `GOPrediction`:

- `identity_nw/sw` — fraction of aligned positions with identical residues.
- `similarity_nw/sw` — fraction of aligned positions with positive BLOSUM62 score.
- `alignment_score_nw/sw` — raw alignment score.
- `gaps_pct_nw/sw` — fraction of the alignment that is gap.
- `alignment_length_nw/sw` — number of aligned columns.
- `length_query / length_ref` — sequence lengths.

5.5.2 Taxonomic Distance

For each (query, reference) pair where taxonomy IDs are available from UniProt metadata, the NCBI taxonomy tree is consulted via ete3 to compute:

- `taxonomic_lca` — the taxon ID of the LCA.
- `taxonomic_distance` — the total edge count between the two taxa through their LCA.
- `taxonomic_common_ancestors` — the number of shared ancestors.
- `taxonomic_relation` — a categorical label: *same_species*, *same_genus*, *same_family*, *same_order*, *same_class*, *same_phylum*, *same_kingdom*, or *distant*.

Taxonomy lookups are memoised with an LRU cache keyed by taxon ID pair to avoid redundant tree traversals across a batch.

5.6 Approximate Nearest-Neighbour Search

The kNN search backend is configurable per prediction job via the `search_backend` payload field. Two backends are supported:

`numpy` Computes exact cosine distances as a matrix product followed by L2 normalisation. This is $O(NQ)$ in both time and memory, but is acceptable for reference sets up to ~100k proteins when the reference embeddings fit in RAM as float16 (approximately 250 MB for 100k $\times$ 1280 dimensions).

`faiss` Uses the FAISS library [13] with a configurable index type. `IVFFlat` partitions the vector space into Voronoi cells and restricts the search to the `nprobe` nearest cells, reducing query time from $O(N)$ to approximately $O(\sqrt{N})$ with negligible recall loss for typical parameter settings.

The pgvector `VECTOR` type is used solely for storage and retrieval; nearest-neighbour queries are never issued via SQL, as pgvector's index performance degrades at the scale of hundreds of thousands of vectors under concurrent load.

5.7 Prediction Export

Prediction results can be exported as a tab-separated file through the endpoint `GET /embeddings/prediction-sets/{id}/predictions.tsv`. The response uses `StreamingResponse` with `yield_per(1000)` to avoid loading the full result set into memory. The exported file contains 27 columns covering protein accessions, GO term, cosine distance, reference evidence code, alignment statistics, and taxonomy fields. Optional query-string filters allow the client to restrict the export by accession, GO aspect (`F/P/C`), or maximum distance threshold.

5.8 Process Management

PROTEA uses a shell script (`scripts/manage.sh`) to manage the set of long-running processes required by the stack. The script starts, stops, and reports the status of nine process roles:

1. FastAPI server (`uvicorn`)
2. Worker: `protea.ping`
3. Worker: `protea.jobs`
4. Worker: `protea.embeddings` (serialised coordinator)
5. $N\times$ Worker: `protea.embeddings.batch` (GPU inference)
6. $N\times$ Worker: `protea.embeddings.write`
7. $N\times$ Worker: `protea.predictions.batch` (kNN)
8. $N\times$ Worker: `protea.predictions.write`
9. Next.js development server

PID files and log files are written under `logs/`. The `manage.sh scale` subcommand adds extra batch workers to a queue without restarting the rest of the stack.

5.9 Frontend

The web interface is a Next.js 19 application using Tailwind CSS v4 for styling. It communicates with the FastAPI backend exclusively through the REST API, enabling the two layers to be deployed independently. The primary workflows exposed by the frontend are:

- Uploading FASTA files to create query sets.
- Triggering and monitoring embedding and prediction jobs.
- Browsing prediction results grouped by protein and GO aspect.
- Downloading predictions as TSV with aspect and distance filters.

5.10 Testing Strategy

The test suite is split into two categories:

Unit tests Run with plain `pytest`. They mock external services (HTTP, RabbitMQ) and use an in-memory SQLite database or minimal fixtures. They cover operation logic, alignment and taxonomy utilities, FASTA parsing, and API router behaviour.

Integration tests Run with `pytest -with-postgres`. The `conftest.py` fixture pulls a `pgvector/pgvector:pg16` Docker image, initialises the schema, and tears down the container after the session. These tests exercise the full round-trip from job submission to database state.

Coverage is enforced at 70 % by `pytest-cov`; the current suite contains 283 passing tests across 17 test files.

Chapter 6

Evaluation

This chapter describes the experimental setup used to assess PROTEA's prediction quality, the baseline methods against which it is compared, the metrics employed, and the results obtained.

6.1 Experimental Setup

All experiments are run on a single workstation equipped with an NVIDIA GPU (16 GB VRAM), 64 GB system RAM, and a PostgreSQL 16 instance storing approximately 570,000 Swiss-Prot proteins. Embeddings are computed with ESM-2 (650M parameter variant) using the `facebook/esm2_t33_650M_UR50D` checkpoint from Hugging Face Transformers, with mean pooling over all residue representations.

The reference annotation set is derived from Swiss-Prot experimental annotations (evidence codes `EXP`, `IDA`, `IMP`, `IGI`, `IEP`, `IPI`) loaded through the `load_quickgo_annotations` operation against the GO release dated 2024-01-17.

6.2 Benchmark Dataset

6.2.1 Construction

To construct a benchmark that avoids evaluation on proteins already used as references, we apply a temporal holdout strategy inspired by CAFA [26]:

1. A snapshot of Swiss-Prot experimental annotations from 2022-01-01 is used as the reference set (hereafter *ref-2022*).

2. Proteins that received their first experimental GO annotation between 2022-01-01 and 2024-01-17 constitute the *test set*; their annotations in the later

snapshot serve as ground truth.

3. Proteins already annotated in ref-2022 are excluded from the test set to prevent trivial retrieval of self-annotations.

This yields approximately *[N]* query proteins with a total of *[M]* experimental GO term annotations distributed across the three aspects (table 6.1).

Table 6.1: Benchmark dataset statistics.

Aspect	**Proteins**	**Annotations**	**Avg. terms / protein**
Molecular Function (MF)	*[N]*	*[M]*	*[avg]*
Biological Process (BP)	*[N]*	*[M]*	*[avg]*
Cellular Component (CC)	*[N]*	*[M]*	*[avg]*
Total	*[N]*	*[M]*	*[avg]*

6.2.2 Sequence Identity Distribution

A critical property of the benchmark is the distribution of maximum sequence identity between each test protein and its nearest neighbour in ref-2022 (as measured by BLAST). Figure 6.1 shows this distribution, which spans the full range from near-identical resequencing artefacts to remote homologues below 30 % identity. We report results stratified by identity bracket to separate the easy (high-identity) and hard (low-identity) regimes.

[Histogram of max BLAST identity — to be added]

Figure 6.1: Distribution of maximum sequence identity between test proteins and their nearest reference neighbour.

6.3 Baseline Methods

PROTEA predictions are compared against three baselines:

BLAST transfer For each test protein, `BLASTP` is run against ref-2022 (E-value $\leq 10^{-3}$); GO terms are transferred from the top hit weighted by bit-score. This represents the standard practice in genome annotation pipelines.

DIAMOND transfer Identical transfer strategy using DIAMOND for alignment, providing a high-throughput variant of BLAST transfer.

Naïve embedding kNN (no features) PROTEA with `compute_alignments=false` and `compute_taxonomy=false`, using cosine distance from ESM-2 embeddings as the sole signal. This ablates the feature-engineering contribution.

6.4 Metrics

We adopt the evaluation protocol and metrics of the CAFA challenge [26]:

$F_{\max}$ The maximum F_1 score over all prediction confidence thresholds $\tau \in [0, 1]$, computed per GO aspect. Precision and recall are defined over the full GO hierarchy using the true path rule.

AUPR Area under the precision–recall curve, summarising performance across all thresholds.

$S_{\min}$ Minimum remaining uncertainty, a semantic distance metric that penalises predictions that are informative but wrong more than uninformative ones [7].

All metrics are computed separately for MF, BP, and CC. Because the cosine distance output by PROTEA is a dissimilarity score (lower is better), we convert it to a confidence score as $c = 1 - d$ before computing threshold-based metrics.

6.5 Results

Table 6.2 reports $F_{\max}$ for all methods across the three GO aspects.

Table 6.3 reports AUPR.

Table 6.2: $F_{\max}$ on the temporal holdout benchmark (higher is better). Best result per column in **bold**.

Method	**MF**	**BP**	**CC**
BLAST transfer	–	–	–
DIAMOND transfer	–	–	–
ESM-2 kNN (no features)	–	–	–
PROTEA (+ alignment)	–	–	–
PROTEA (+ align + tax.)	–	–	–

Table 6.3: AUPR on the temporal holdout benchmark (higher is better).

Method	**MF**	**BP**	**CC**
BLAST transfer	–	–	–
DIAMOND transfer	–	–	–
ESM-2 kNN (no features)	–	–	–
PROTEA (+ alignment)	–	–	–
PROTEA (+ align + tax.)	–	–	–

6.5.1 Performance by Sequence Identity Bracket

Figure 6.2 disaggregates $F_{\max}$ (MF aspect) by the maximum sequence identity of each test protein to its nearest reference. The embedding approach is expected to maintain predictive power in the 10–30 % identity range where alignment-based methods degrade most sharply.

[$F_{\max}$ vs. identity bracket plot — to be added]

Figure 6.2: $F_{\max}$ (Molecular Function) stratified by maximum sequence identity to the reference set.

6.6 Ablation Study

To quantify the individual contribution of each feature-engineering signal, we run four configurations of PROTEA on the full test set:

1. Cosine distance only (baseline kNN).
2. + NW identity.
3. + NW identity + SW identity.
4. + NW/SW identity + taxonomic distance.

Results are reported in table 6.4. The primary comparison of interest is the gain from adding alignment over embedding similarity alone, and the incremental gain from adding taxonomy.

Table 6.4: Ablation study: $F_{\max}$ (MF) for progressive feature addition.

Configuration	$F_{\max}$ **(MF)**
Cosine only	–
+ NW identity	–
+ NW + SW identity	–
+ NW + SW + taxonomic distance	–

6.7 Throughput and Latency

In addition to prediction quality, we report end-to-end throughput for a representative workload of 10,000 query proteins against the full Swiss-Prot reference:

- **Embedding computation**: *[X]* sequences/second on a single GPU worker (ESM-2 650M, batch size 64, `cuda`).
- **kNN prediction (NumPy)**: *[X]* queries/second.
- **kNN prediction (FAISS IVFFlat)**: *[X]* queries/second.
- **Total wall-clock time** (embed + predict + write): *[X]* min.

6.8 Discussion

The results are interpreted in light of the three research questions posed in section 1.2.

RQ1 — Whether embedding-based kNN matches or exceeds alignment-based methods at low identity. The stratified results (fig. 6.2) are expected to show that ESM-2 kNN is competitive with BLAST transfer in the high-identity regime and substantially better below 30 % identity, consistent with findings in Littmann et al. [17].

RQ2 — Whether alignment and taxonomy features improve ranking. The ablation study (table 6.4) will quantify this. The working hypothesis is that NW identity provides the largest incremental gain over cosine distance alone, while taxonomic distance provides a smaller but complementary signal, particularly for proteins from distantly related lineages.

RQ3 — Whether the versioned pipeline design achieves reproducibility. This is validated structurally: every prediction in the database is linked to an immutable `EmbeddingConfig` UUID, a specific `AnnotationSet`, and a specific `OntologySnapshot`. Re-running the same job payload against the same database state reproduces bit-identical results.

6.8.1 Limitations

Several caveats apply to the evaluation:

- The temporal holdout strategy reduces contamination but does not eliminate it entirely: some proteins in the test set may share high sequence identity with ref-2022 proteins, making their annotations trivially predictable by any method.

- ESM-2 650M was chosen for computational tractability; larger variants (3B, 15B) would likely improve embedding quality at significantly higher GPU cost.

- The evaluation considers only annotation transfer from Swiss-Prot; the QuickGO annotation set (which includes IEA annotations) is not evaluated as a reference because IEA annotations are themselves computationally derived.

Chapter 7

Conclusion

7.1 Summary

This thesis presented PROTEA, a distributed platform for scalable, reproducible protein functional annotation using gene ontology terms. The work was motivated by the widening gap between the rate of protein sequence discovery and the capacity of experimental characterisation, and by the lack of existing platforms that combine modern protein language model embeddings with reproducible versioned pipelines.

The main contributions are:

An open-source platform implementing the full annotation workflow from FASTA upload to structured GO predictions, backed by a REST API and a web interface. The system handles reference sets of hundreds of thousands of proteins and is horizontally scalable through queue-based worker pools.

A versioned relational data model in which every prediction is permanently linked to the exact ontology release, annotation set, and embedding configuration used to produce it. This enables retrospective re-evaluation as databases and ontologies are updated, addressing a gap identified in chapter 3.

A feature-engineering module that augments embedding cosine similarity with Needleman–Wunsch and Smith–Waterman alignment statistics and NCBI-taxonomy-based distance metrics. The ablation study in chapter 6 quantifies the contribution of each signal.

An empirical evaluation on a temporal holdout of Swiss-Prot, comparing PROTEA against BLAST and DIAMOND transfer baselines and stratifying performance by sequence identity bracket to highlight the relative advantage of embedding-based methods at low identity.

The biological context — protein sequences, the Gene Ontology, annotation evidence standards, and taxonomic classification — was treated as the primary serious instance of the system, motivating every major design decision. The engineering challenges addressed (GPU memory management, approximate nearest-neighbour search, streaming exports, process-level reference caches) are direct consequences of operating at biological database scale.

7.2 Limitations

Several limitations constrain the current version of PROTEA:

Single-model embeddings. PROTEA currently supports one embedding model per `EmbeddingConfig`. Ensembling across multiple pLMs (e.g. ESM-2 + ProtT5) would require either multi-column vector storage or a separate embedding per model, which increases disk and memory requirements.

No structural information. Protein tertiary structure is a powerful predictor of function. AlphaFold2 has made structural predictions available at proteome scale; incorporating structure-based embeddings (e.g. from ESM-IF or Foldseek) into the pipeline is a natural extension.

Flat annotation transfer. The current kNN transfer takes GO terms from the k nearest neighbours without propagating up the GO DAG or calibrating confidence scores beyond cosine distance. A post-processing step that propagates annotations and applies isotonic regression could substantially improve $F_{\max}$.

Single-node deployment. The current architecture runs on a single server managed by `manage.sh`. A Kubernetes-based deployment would enable auto-scaling and cloud portability.

Benchmark coverage. The temporal holdout benchmark covers proteins that received their first experimental annotation in a fixed two-year window. This over-represents proteins that are easy to annotate experimentally and may under-represent remote homologues and orphan proteins.

7.3 Future Work

Several directions are identified for future development:

Structure-aware embeddings. Integrating structure-based representations alongside sequence embeddings would extend PROTEA's utility to the >200 million AlphaFold Database entries whose functional annotation is currently sparse. A separate `EmbeddingConfig` for structural vectors could be stored alongside sequence embeddings and combined at prediction time.

Ontology-aware scoring. Replacing flat majority vote with a GO-aware scoring scheme — for instance, weighting neighbour annotations by their information content within the DAG, or using a hierarchical classifier trained on the retrieved neighbourhood — is expected to reduce semantic distance (S_{min}) without sacrificing recall.

Active learning integration. PROTEA's versioned predictions could serve as a scaffold for active learning: proteins with high embedding distance to all reference neighbours (high uncertainty) could be prioritised for experimental characterisation, closing the annotation loop.

Cloud-native deployment. Containerising each worker role and deploying on a managed Kubernetes cluster would enable PROTEA to serve multiple research groups concurrently, with auto-scaling of GPU batch workers as demand fluctuates.

Benchmark expansion. Extending the evaluation to the CAFA5 target set and to whole-proteome annotation tasks for model organisms would provide a broader picture of PROTEA's generalisation and facilitate direct comparison with CAFA participants.

Explainability. Providing users with alignment visualisations and ontology subgraph views for each prediction — showing why a specific reference was retrieved and which evidence supports the transferred term — would improve trust and enable biologists to validate predictions without re-running alignments manually.

PROTEA is released as open-source software. The codebase, schema migrations, and operational documentation are publicly available at *[repository URL]*.

Bibliography

[1] Stephen F. Altschul et al. “Basic local alignment search tool”. In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. DOI: 10.1016/S0022-2836(05)80360-2.

[2] Michael Ashburner et al. “Gene Ontology: tool for the unification of biology”. In: *Nature Genetics* 25.1 (2000), pp. 25–29. DOI: 10.1038/75556.

[3] Helen M. Berman et al. “The Protein Data Bank”. In: *Nucleic Acids Research* 28.1 (2000), pp. 235–242. DOI: 10.1093/nar/28.1.235.

[4] Matthias Blum et al. “InterPro in 2021 – improving sequence-based predictions and integrating AlphaFold2 structures”. In: *Nucleic Acids Research* 49.D1 (2021), pp. D344–D354. DOI: 10.1093/nar/gkab994.

[5] Benjamin Buchfels et al. “DIAMOND protein aligner”. In: *Nature Methods* 18 (2021), pp. 366–368. DOI: 10.1038/s41592-021-01101-x.

[6] Carlos P. Cantalapiedra et al. “eggNOG-mapper v2: functional annotation, orthology assignments, and domain prediction at the metagenomic scale”. In: *Molecular Biology and Evolution* 38.12 (2021), pp. 5825–5829. DOI: 10.1093/molbev/msab293.

[7] Wyatt T. Clark and Predrag Radivojac. “Information-theoretic evaluation of predicted ontological annotations”. In: *Bioinformatics* 29.13 (2013), pp. i53–i61. DOI: 10.1093/bioinformatics/btt228.

[8] Sean R. Eddy. “Accelerated profile HMM searches”. In: *PLOS Computational Biology* 7.10 (2011). DOI: 10.1371/journal.pcbi.1002195.

[9] Ahmed Elnaggar et al. “ProtTrans: Towards Cracking the Language of Life’s Code through Self-Supervised Deep Learning and High Performance Computing”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.10 (2022), pp. 7112–7127. DOI: 10.1109/TPAMI.2021.3095381.

[10] Marco Falda et al. "Argot2: a large scale function prediction tool relying on semantic similarity of weighted Gene Ontology terms". In: *BMC Bioinformatics* 13.Suppl 4 (2012), S14. DOI: 10.1186/1471-2105-13-S4-S14.

[11] Gene Ontology Consortium et al. "The Gene Ontology knowledgebase in 2023". In: *Genetics* 224.1 (2023). DOI: 10.1093/genetics/iyad031.

[12] Steven Henikoff and Jorja G. Henikoff. "Amino acid substitution matrices from protein blocks". In: *Proceedings of the National Academy of Sciences* 89.22 (1992), pp. 10915–10919. DOI: 10.1073/pnas.89.22.10915.

[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *IEEE Transactions on Big Data* 7.3 (2021), pp. 535–547. DOI: 10.1109/TBDATA.2019.2921572.

[14] Maxat Kulmanov and Robert Hoehndorf. "DeepGOPlus: improved protein function prediction from sequence". In: *Bioinformatics* 36.2 (2020), pp. 422–429. DOI: 10.1093/bioinformatics/btz595.

[15] Maxat Kulmanov, Mohammed Asif Khan, and Robert Hoehndorf. "DeepGO: predicting protein functions from sequence and interactions using a deep ontology-aware classifier". In: *Bioinformatics* 34.4 (2018), pp. 660–668. DOI: 10.1093/bioinformatics/btx624.

[16] Zeming Lin et al. "Evolutionary-scale prediction of atomic-level protein structure with a language model". In: *Science* 379.6637 (2023), pp. 1123–1130. DOI: 10.1126/science.ade2574.

[17] Maria Littmann et al. "Embeddings from deep learning transfer GO annotations beyond homology". In: *Scientific Reports* 11 (2021), p. 1160. DOI: 10.1038/s41598-020-80786-0.

[18] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453. DOI: 10.1016/0022-2836(70)90057-4.

[19] Jong Park, Kevin Karplus, et al. "Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods". In: *Journal of Molecular Biology* 284.4 (1998), pp. 1201–1210. DOI: 10.1006/jmbi.1998.2221.

[20] Burkhard Rost. "Twilight zone of protein sequence alignments". In: *Protein Engineering* 12.2 (1999), pp. 85–94. DOI: 10.1093/protein/12.2.85.

[21] Conrad L. Schoch et al. "NCBI Taxonomy: a comprehensive update on curation, resources and tools". In: *Database* 2020 (2020). DOI: 10.1093/database/baaa062.

[22] Temple F. Smith and Michael S. Waterman. "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5.

[23] Petri Törönen et al. "PANNZER2: a rapid functional annotation webserver". In: *Nucleic Acids Research* 46.W1 (2018), W84–W88. DOI: 10.1093/nar/gky350.

[24] UniProt Consortium. "UniProt: the Universal Protein Knowledgebase in 2023". In: *Nucleic Acids Research* 51.D1 (2023), pp. D523–D531. DOI: 10.1093/nar/gkac1052.

[25] Ronghui You et al. "NetGO: improving large-scale protein function prediction with massive network information". In: *Nucleic Acids Research* 47.W1 (2019), W379–W387. DOI: 10.1093/nar/gkz370.

[26] Naihui Zhou et al. "The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens". In: *Genome Biology* 20.1 (2019), p. 244. DOI: 10.1186/s13059-019-1835-8.

Appendix A

REST API Reference

This appendix documents the public endpoints of the PROTEA REST API. All endpoints are prefixed with the base URL of the FastAPI server (default: `http://127.0.0.1:8000`).

A.1 Jobs

`POST /jobs` Submit a new job. The request body must contain `operation` (string) and `payload` (JSON object). Returns the created `Job` row.

`GET /jobs` List jobs, optionally filtered by `status` and `operation` query parameters.

`GET /jobs/{id}` Retrieve a single job by UUID.

`GET /jobs/{id}/events` Stream the structured event log for a job as newline-delimited JSON.

`POST /jobs/{id}/cancel` Request cancellation of a queued or running job.

A.2 Proteins and Sequences

`POST /proteins/insert` Enqueue an `insert_proteins` job for a list of UniProt accessions or a taxonomy query.

`POST /proteins/metadata` Enqueue a `fetch_uniprot_metadata` job to refresh functional annotation data.

`GET /proteins` List proteins stored in the database with optional accession filter.

A.3 Query Sets

`POST /query-sets` Upload a FASTA file. Creates a `QuerySet` and its `QuerySetEntry` rows; upserts novel sequences and proteins. Returns the created `QuerySet` UUID.

`GET /query-sets` List all query sets.

`GET /query-sets/{id}` Retrieve a query set with its entries.

`DELETE /query-sets/{id}` Delete a query set and its entries.

A.4 Annotations

`POST /annotations/ontology-snapshot` Enqueue a `load_ontology_snapshot` job for a given OBO URL.

`POST /annotations/annotation-set/goa` Enqueue a `load_goa_annotations` job for a local GAF file path.

`POST /annotations/annotation-set/quickgo` Enqueue a `load_quickgo_annotations` job with optional evidence code filter.

`GET /annotations/ontology-snapshots` List loaded ontology snapshots.

`GET /annotations/annotation-sets` List loaded annotation sets.

A.5 Embeddings and Predictions

`POST /embeddings/configs` Create a new `EmbeddingConfig`.

`GET /embeddings/configs` List embedding configurations.

`POST /embeddings/compute` Enqueue a `compute_embeddings` coordinator job.

`POST /embeddings/predict` Enqueue a `predict_go_terms` coordinator job.

`GET /embeddings/prediction-sets` List prediction sets.

`GET /embeddings/prediction-sets/{id}` Retrieve a prediction set with summary statistics.

`GET /embeddings/prediction-sets/{id}/predictions.tsv` Stream predictions as a 27-column TSV file. Optional query parameters: `accession`, `aspect` (`F`/`P`/`C`), `max_distance`.