Universidad *[University Name]*

Programa de Doctorado en *[Programme Name]*

Departamento de *[Department Name]*

PROTEA: A Framework for Scalable Protein Functional Annotation

---

Doctoral Thesis

submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

Author
Francisco Percan

Thesis Director
[Director Name]

Co-Director
[Co-Director 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. The gap is widening: large-scale protein language models (PLMs) and structural predictors generate dense representations of every UniProt sequence in days, yet the infrastructure required to turn those representations into auditable, reproducible Gene Ontology predictions is fragmented across notebooks, prototype scripts and one-off benchmarks.

This thesis presents **PROTEA** (*PROtein funcTional annotation framEwork and plAtform*), an open-source distributed platform that automates the full pipeline from raw protein sequence to structured Gene Ontology predictions, and provides an extensible foundation on top of which third parties can plug in new PLMs, annotation sources, and ranking methods. PROTEA combines protein language model embeddings (eight backends evaluated, including ESM-2 in three sizes, ESM-C, ProstT5, ProtT5, Ankh and ESM-3 component encoder), approximate k-nearest-neighbour search over 527,000 reviewed UniProt sequences, and evidence transfer from versioned annotation sets to produce ranked, interpretable predictions. A versioned relational data model links every prediction to the exact ontology release, annotation set, embedding configuration and code commit that produced it, enabling exact replay of any past run.

To improve prediction quality beyond embedding cosine similarity, PROTEA incorporates a plugin-based feature engineering layer that computes Needleman–Wunsch and Smith–Waterman alignment statistics, NCBI-taxonomy-based distance metrics, ancestry embeddings of the GO directed acyclic graph, and per-PLM PCA projections, yielding a schema of approximately 52 features per candidate annotation. A LightGBM re-ranker is trained on these features using a temporal holdout protocol spanning twelve consecutive GOA releases, with Information Accretion weighting, per-category models (No-Knowledge, Limited-Knowledge, Prior-Knowledge), and

selective application that falls back to the embedding-only baseline when re-ranking would degrade performance.

Beyond the prediction method, the thesis describes the engineering pillars that make the platform deployable across heterogeneous environments: a seven-repository plugin architecture discovered via Python entry points, externalised configuration via `pydantic-settings`, OpenTelemetry-based distributed tracing, multi-target container packaging covering development, cloud, HPC (Apptainer) and airgapped environments, and an operational narrative model that makes every Job and `ExperimentRun` self-describing.

Evaluation follows the CAFA temporal protocol on held-out GOA splits and reports the performance of the canonical pipeline across the eight PLM backends and three category slices, contextualises the impact of each feature family through ablation studies, and documents the methodological pivots that shaped the canonical configuration. Three pre-built containers are submitted to the LAFA benchmark to demonstrate adoption-ready deployment.

Keywords: protein function prediction, gene ontology, protein language models, sequence embeddings, nearest-neighbour search, learning-to-rank, LightGBM, distributed systems, reproducibility, container deployment, bioinformatics infrastructure.

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, 32

GAF GO Annotation File. 26

GO Gene Ontology. 1, 2, 5

HMM Hidden Markov Model. 1, 10

kNN k-Nearest Neighbours. 1, 31, 34

LCA Lowest Common Ancestor. 6, 27

MF Molecular Function. 5

NW Needleman–Wunsch. 6, 21, 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, 21, 26

Swiss-Prot Swiss Protein database (reviewed section of UniProt). 4

TrEMBL Translated EMBL Nucleotide Sequence Data Library. 4

UniProt Universal Protein Resource. 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, Universal Protein Resource (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. Over the last five years, however, Protein Language Models (pLMs) trained on hundreds of millions of sequences have produced dense vector representations that capture structural and functional signal well beyond what classical alignment can detect, and a generation of structure-conditioned models (ESM-3, ProstT5) has begun to integrate sequence and structure into a single representation. Each new PLM unlocks a new operating point in the accuracy/throughput trade-off, but also introduces a new dependency, a new tokeniser, a new GPU memory footprint, and a new failure mode. Comparing PLMs at scale, on a fair benchmark, with reproducible numbers, is itself a research engineering problem.

PROTEA was designed to address both faces of the problem at once: as a method, it operationalises embedding-based annotation transfer with selective learning-to-rank re-finement; as a platform, it provides the extensible foundation that lets new backends, annotation sources, and ranking methods be plugged in without ad-hoc

glue. The platform is deliberately built around the assumption that the canonical pipeline of today will be obsolete tomorrow. What must remain stable is the contract between layers, the trace of every run, and the pathway from a FASTA file to a ranked, interpretable Gene Ontology prediction.

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 Gene Ontology (GO) term prediction, particularly at low sequence identity, and how does the answer change as new PLMs replace older ones?

RQ2. Do pairwise alignment statistics, taxonomic distance, ontology-graph ancestry features, and per-PLM PCA projections provide complementary signal that improves the ranking of candidate annotations once the embedding baseline is already strong?

RQ3. How can a reproducible, scalable annotation pipeline be designed so that researchers can re-run analyses as ontologies and databases are updated, and so that the pipeline itself can be deployed in development, cloud, on-premise HPC, and airgapped environments without code branching?

RQ4. How can the journey of a research platform, with its dead ends and pivots, be captured operationally so that the canonical pipeline is justified by an auditable trace rather than by post-hoc rationalisation?

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 and selective learning-to-rank re-ranking.
- A **plugin architecture** based on Python `entry_points`, organised across seven code repositories, that lets new annotation sources, runners and PLM backends be added without modifying the core platform.

- A **versioned data model** for ontology snapshots, annotation sets, embedding configurations, datasets, re-ranker models and experiment runs, with provenance complete enough to exactly replay any past prediction.
- A **benchmark of eight PLM backends** (ESM-2 in three sizes, ESM-C, ProstT5, ProtT5, Ankh, ESM-3 component encoder) on a common evaluation surface, with per-aspect, per-category and per-backend numbers.
- A **feature engineering layer** that augments embedding similarity with Needleman–Wunsch / Smith–Waterman alignment scores, NCBI-taxonomy-based distance metrics, GO-graph ancestry embeddings, and per-PLM PCA projections, organised as a registry of approximately 52 features.
- A **selective learning-to-rank re-ranker** trained per category and per aspect, that falls back to the embedding-only baseline whenever re-ranking would degrade performance for the current cell.
- A **multi-target deployment story** (development `docker compose`, cloud Helm/Compose, HPC Apptainer, airgapped bundle) covering the same canonical pipeline without code branching.
- An **operational narrative model** (`Job.findings`, `ExperimentRun`, comments) that captures the why of every run and provides the audit trace from which the evaluation chapter is distilled.
- An **empirical evaluation** comparing PROTEA predictions against baseline methods on held-out Gene Ontology Annotation (GOA) splits following the CAFA temporal protocol, complemented by three submissions to the LAFA public benchmark.

1.4 Thesis Structure

The remainder of this document is organised as follows. Chapter 2 introduces the biological concepts underpinning the work, including the modern landscape of protein language models and their structural extensions. Chapter 3 surveys existing computational approaches to protein function prediction up to 2025, with attention to the most recent embedding-based and graph-based methods. Chapter 4 presents the system architecture of PROTEA: design goals, the operation abstraction, plugin contracts, the job lifecycle, the queue topology, the data model, the embedding and prediction pipelines, and the public REST API. Chapter 5 describes how the design

materialises across the seven repositories, covering the technology stack, the database schema and migrations, the operations catalogue, feature engineering, approximate nearest-neighbour search, the reranker promotion pipeline, the frontend, and the testing strategy. Chapter 6 reports the experimental results on the eight-PLM benchmark, defines the NK/LK/PK evaluation categories and the metrics, and walks through a sequence of experiments from KNN baselines to the LightGBM reranker, including a comparative study against eggNOG-mapper. Chapter 7 summarises the findings, answers the research questions, discusses limitations, and outlines future directions including the LAFA submissions and the post-defense agenda.

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 Protein database (reviewed section of UniProt) (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 trained with a masked language modelling objective on hundreds of millions of sequences develop internal representations that capture evolutionary, structural, and functional information without explicit supervision.

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, which is what makes embeddings attractive for annotation transfer.

2.5.1 The 2020–2024 Generation: Sequence-Only Encoders

The first generation of practical pLMs for function prediction was based on Transformer encoders trained with a masked language modelling objective on sequence-only corpora.

ESM-1b [**rives2021esm1b**] and the **ESM-2** family [16] (Meta AI) span eight model sizes from 8M to 15B parameters, all trained on UniRef50 / UniRef90.

Each scale increment is accompanied by a measurable gain on structure prediction and remote-homology detection benchmarks. PROTEA evaluates ESM-2 at three operating points (650M, 3B, 15B) to characterise the accuracy/throughput trade-off.

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

Ankh [**elnaggar2023ankh**] is an encoder–decoder model trained on UniRef50 with a focused objective and aggressive vocabulary tuning. It reports state-of-the-art accuracy on several downstream tasks at a fraction of the parameter count of the largest ESM-2 variants.

2.5.2 The 2024–2025 Generation: Structure-Aware and Distilled Models

Three trends shape the most recent generation: integration of structural signal during training, further parameter efficiency, and sequence/structure multi-modal representations.

ProstT5 [**heinzinger2023prostt5**] extends ProtT5 with a bilingual sequence-to-3Di mapping (3Di is the Foldseek structural alphabet derived from predicted three-dimensional structures), letting the model encode structural information without an explicit 3D input. The same encoder serves both sequence-only and structure-aware downstream tasks.

ESM-3 [**hayes2024esm3**] is a multi-modal generative model over sequence, structure and function tokens, reaching parameter scales up to 98B. For embedding-based downstream applications PROTEA uses the lighter component encoder (ESM-3c), which provides per-residue and per-protein embeddings comparable in interface to the ESM-2 family.

ESM-C [**esm2024esmc**] (released in late 2024 by EvolutionaryScale) is a family of efficient sequence-only encoders distilled from larger ESM-3 variants. ESM-C 300M and 600M deliver embeddings competitive with ESM-2 3B at a fraction of the inference cost, making it the smallest backend at which PROTEA achieves strong CAFA-style performance.

2.5.3 Inference Cost and the Operating-Point Question

Each PLM occupies a distinct operating point on the accuracy/throughput trade-off. ESM-2 15B produces the strongest embeddings but requires several days on

a single A100 to embed 500,000 sequences; ESM-C 300M produces embeddings of comparable downstream quality on the same hardware in under a day. Choosing a backend is therefore a deployment decision more than a modelling one, and a platform that operationalises PLM-based annotation at scale must abstract the choice cleanly. Chapter 4 describes how PROTEA implements that abstraction through the `EmbeddingBackend` plugin contract.

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 [**rives2021esm1b**] and the **ESM-2** family [16] (Meta AI) are transformer encoders trained on UniRef50/90, scaling from 8M to 15B parameters. The larger variants generate embeddings that outperform BLAST-based transfer at low-identity regimes. ESM-2 embeddings have served as the de-facto baseline representation in the function prediction literature since 2023.

ProtTrans [9] provides a suite of encoder models (ProtBERT, ProtAlbert, 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.

Ankh [**elnaggar2023ankh**] is an encoder–decoder model that matches or exceeds the ESM-2 3B baseline on multiple downstream tasks while using a fraction of the parameters, achieved through aggressive vocabulary tuning and a re-thought training recipe.

The 2024–2025 generation moves in three complementary directions. First, **ProstT5** [**heinzinger2023prostt5**] introduces structural awareness without requiring explicit 3D input by training a sequence-to-3Di translation on top of ProtT5, where 3Di is the structural alphabet introduced by Foldseek [**vankempen2024foldseek**]. Second, **ESM-3** [**hayes2024esm3**] unifies sequence, structure and function as multi-modal tokens within a single generative model up to 98B parameters; for embedding-based downstream applications, the lighter component encoder (ESM-3c) is the practical choice. Third, **ESM-C** [**esm2024esmc**] delivers efficient sequence-only encoders distilled from ESM-3, with the 300M and 600M variants reaching ESM-2 3B performance at a fraction of the inference cost. PROTEA evaluates all of these as plug-in backends within a single benchmarking framework.

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 Ontology-Aware Representations

The Gene Ontology is a structured DAG, and several recent methods exploit that structure to inject ontology-aware signal into the prediction pipeline, rather than treating each GO term as an independent label.

anc2vec [**edera2022anc2vec**] learns a distributed representation of each GO term from its ancestor multiset under the *is-a* and *part-of* relations. Two terms with overlapping lineage have similar anc2vec vectors, providing a feature with which a downstream classifier can penalise predictions that contradict the true-path rule. PROTEA uses anc2vec features as an input to its re-ranker.

GeOKG [**geokg2025**] (Bioinformatics, 2025) extends this idea by embedding the GO graph in a multi-curvature space combining hyperbolic and Euclidean geometry, motivated by the observation that the GO DAG has both tree-like and grid-like substructure that no single-geometry embedding captures well. GeOKG-style embeddings are evaluated as a possible replacement for anc2vec in chapter 6.

Graph neural networks over the GO DAG have been investigated for end-to-end prediction [**kulmanov2022deepgozero**]. They are an attractive direction for future work but lie outside the scope of the canonical PROTEA pipeline; chapter 7 discusses them in the context of the PROTEA-DL pilot.

3.6 Benchmarks and Evaluation Standards

3.6.1 CAFA

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.

3.6.2 Live Continuous Benchmarks: LAFA

The CAFA cycle releases a frozen evaluation snapshot every two to three years. A complementary movement towards always-on, container-based benchmarks has emerged with **LAFA** [**lafa2024**] (Live Assessment of Functional Annotation, `functionbench.net`). Participants submit a Docker container that consumes a FASTA file and emits predictions in a standardised format; the benchmark host runs the container against a periodically updated held-out set and publishes results. LAFA lowers the barrier to public evaluation: any laboratory with a working method can submit without joining a formal challenge cycle. Three pre-built PROTEA configurations are submitted to LAFA in chapter 7.

3.7 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. PROTEA addresses this gap directly.

3.8 Summary and Positioning of PROTEA

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

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.

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	✓	✓	✓	✓

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.

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.

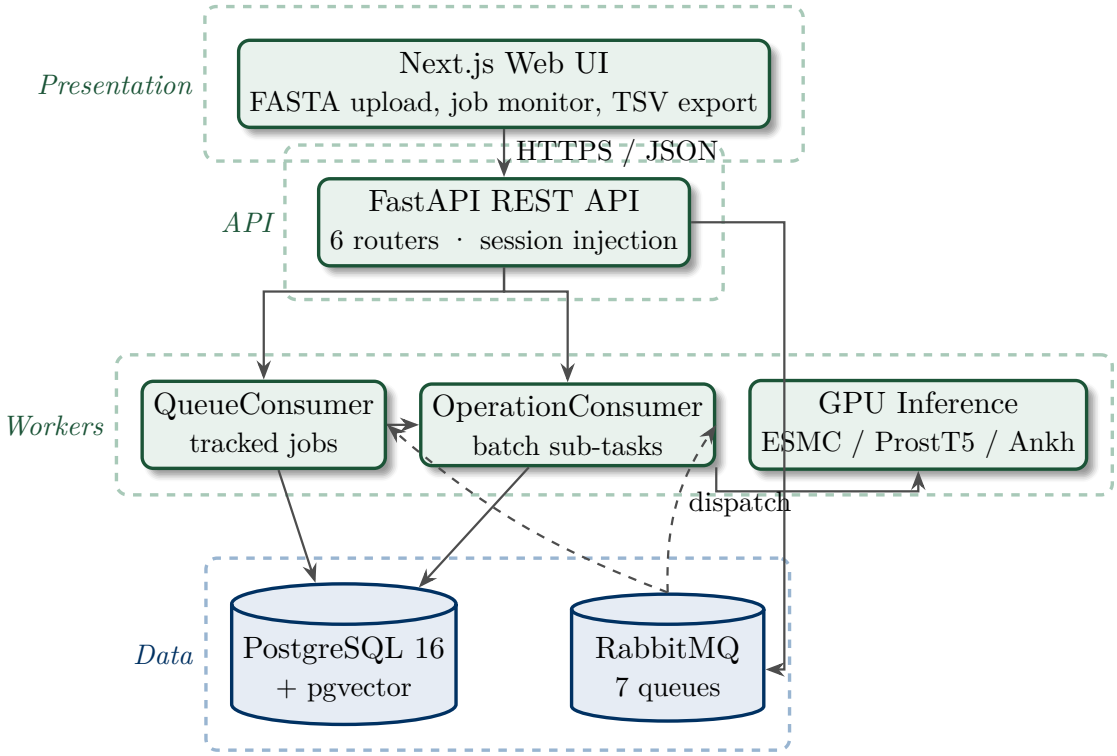

Figure 4.1: High-level architecture of PROTEA. The four horizontal layers (presentation, API, workers, data) communicate exclusively through well-defined interfaces: HTTPS/JSON between the browser and the API, RabbitMQ for task dispatch, and PostgreSQL as the single source of truth. GPU inference is isolated in dedicated workers so that compute capacity can be scaled independently of API throughput.

4.4 Plugin Contracts and the Contracts Package

Operations describe the platform's internal vocabulary, but PROTEA also has to evolve along an orthogonal axis: a new protein language model becomes available, a new annotation source needs to be ingested, a different training algorithm warrants evaluation. If every such addition required edits to the core platform repository, the requirement R3 (separation of concerns) of section 4.1 would be undermined: domain logic and extension points would tangle, third parties could not contribute without forking the platform, and the cadence at which new capabilities can be shipped would be set by the slowest moving piece. The plugin layer answers that pressure.

4.4.1 The Contracts Package

A small Python package, `protea-contracts`, holds the abstract base classes and shared schemas that define the extension surface. The package is deliberately minimal: it depends only on `pydantic`, `numpy` and `pyarrow`, and is forbidden from importing `sqlalchemy`, `fastapi`, or anything from `protea-core`. Plugin repositories install `protea-contracts` without dragging the platform stack along; the inference path likewise consumes the package without paying for the worker, queue and ORM machinery. Three direct consequences follow.

First, plugin repositories can be developed, tested and released independently of `protea-core`. The four backends shipped today (ESM-2 in three sizes, ESM-C, ProstT5, ProtT5, Ankh) live in a separate repository, are installable as `pip install protea-backends[<extra>]`, and ship per-backend Poetry extras that pull only the heavy machine-learning libraries the chosen backend actually needs.

Second, the inference path can ship as a standalone artefact (the `protea-method` package, in extraction at the time of writing) without forcing downstream consumers into the platform's operational shape. The same code that PROTEA runs inside its workers powers the LAFA benchmark submissions described in chapter 7, and any laboratory that wants to deploy a frozen model receives only the dozen megabytes of inference logic.

Third, the contracts themselves become a SemVer-versioned public artefact. Adding a feature to the canonical schema bumps a minor; renaming an ABC method bumps a major. Consumers pin a known-good range and the platform team controls the breakage budget.

4.4.2 The Four Plugin Layers

Table 4.1 summarises the four extension points. Three of them are discovered through Python's `importlib.metadata.entry_points` mechanism: at startup the platform queries each named group, loads the plugin instances, and registers them in an in-process map analogous to the `OperationRegistry`. The fourth, the feature registry, is collected in-process from a fixed list of family modules within `protea-core` itself; it does not use entry points because the per-candidate features have to participate in a single, deterministic ordering for the schema fingerprint described below to be stable.

Table 4.1: The four plugin extension points of PROTEA.

Extension	ABC	Discovery group	Examples
Annotation sources	`AnnotationSource`	`protea.sources`	GOA, QuickGO, UniProt
Embedding backends	`EmbeddingBackend`	`protea.backends`	ESM, T5, Ankh, ESM-C
Experiment runners	`ExperimentRunner`	`protea.runners`	KNN, baseline, LightGBM
Per-candidate features	`FeatureRegistry`	in-process registry	alignment, taxonomy, anc2vec, emb-PCA

Each plugin is a Python module that subclasses the relevant ABC, sets a class attribute `name` matching the entry-point name, and exposes a module-level instance `plugin = MyPlugin()`. Heavy dependencies are imported lazily inside the methods that need them, never at module top. This last detail is load-bearing: `protea-core` startup queries the entry points and instantiates every discovered plugin, which would otherwise pay the import cost of `torch`, `transformers` and `esm` on every container start regardless of which backends are in use.

4.4.3 Discovery and Dispatch

Replacing a hard-coded `if/elif` chain with a registry lookup is a mechanical change with two non-obvious benefits. First, a missing or misspelt plugin name produces a single, readable error (`KeyError("unknown backend: ...")`) rather than the generic fall-through that the chain would have produced. Second, plugins added to the deployment after the binary was packaged become available without a rebuild: the platform image and the plugin packages are decoupled.

A sanity check guards the boundary: the `name` attribute on the loaded plugin instance must equal the entry-point name as declared in the plugin repository's `pyproject.toml`. A mismatch raises at startup rather than producing silent misrouting later. This is the kind of error that becomes invisible if it goes unchecked, since the misrouted plugin still implements the ABC correctly; the platform would happily call it, just under the wrong dispatch key.

4.4.4 Schema Drift and the Canonical Fingerprint

The most operationally consequential schema in `protea-contracts` is the canonical feature list of the re-ranker. Each candidate annotation in a `GOPrediction` row carries a fixed sequence of features (embedding distance, alignment statistics, taxonomic distance, ontology ancestry vectors, per-PLM PCA projections, annotation metadata). The ordering and the names matter: a LightGBM booster trained against feature `[a, b, c]` will silently apply learned weights to the wrong columns if it is later loaded against feature `[a, c, b]`. The output looks plausible, the metrics drift downward by a margin that is indistinguishable from ordinary model variance, and the bug remains invisible until something independent forces an audit.

To prevent that, the package exports a function `compute_schema_sha(features)` that reduces a list of feature names to a stable digest. The re-ranker registry stores the digest as `feature_schema_sha` on every `RerankerModel` row. At inference time, the live pipeline computes the digest from the active registry and refuses to apply a booster whose stored digest does not match. Adding a feature to the canonical list deliberately changes the digest: every booster trained against the old list becomes ineligible and must be retrained, which is the correct behaviour because the new feature has shifted the column space the booster understood.

The motivation for promoting this helper out of `protea-core` and into `protea-contracts` was not theoretical. During the 2026-05-01 evaluation campaign, an experimental run that should have been reproducible deviated by approximately three Fmax points across multiple ontology aspects. The eventual diagnosis traced the discrepancy to two parallel definitions of `compute_schema_sha`: one in the offline training repository (the *lab*), one inside `protea-core`. Both functions were correct in isolation, but a refactor to one had not propagated to the other, and the resulting digests differed for the same input feature list. Boosters registered through one code path were silently being scored under the other.

The fix combined three actions. First, the canonical implementation landed in `protea-contracts`, with a golden test pinning its output to a known digest. Second, the lab and `protea-core` were both amended to import the helper from the contracts package; their previously shadowed copies were deleted. Third, an Alembic migration introduced a parallel column `feature_schema_sha_v2` on the `RerankerModel` and `Dataset` tables, populated by a backfill script that recomputed each row's digest using the canonical helper. The legacy column is retained until the audit window closes; new registrations write only the v2 column. The episode is recorded in ADR D10.

4.4.5 Why a Separate Package, Not Just a Module

The contracts could in principle have lived inside `protea-core` as a designated public sub-module. Three considerations argued against that. The first is the inference shipping question above: the offline trainer and the LAFA submission containers consume contracts but cannot afford the platform stack as a transitive dependency. The second is the SemVer envelope: a sub-module's version moves with the platform, but the contracts evolve at a slower cadence and downstream consumers benefit from being able to pin them independently. The third is the schema drift incident itself, which would have been impossible if there had been a single canonical location for `compute_schema_sha` from the start. The discipline of packaging is the discipline of refusing to let two implementations drift apart in the first place.

4.5 Job Lifecycle and Worker Patterns

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

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 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.

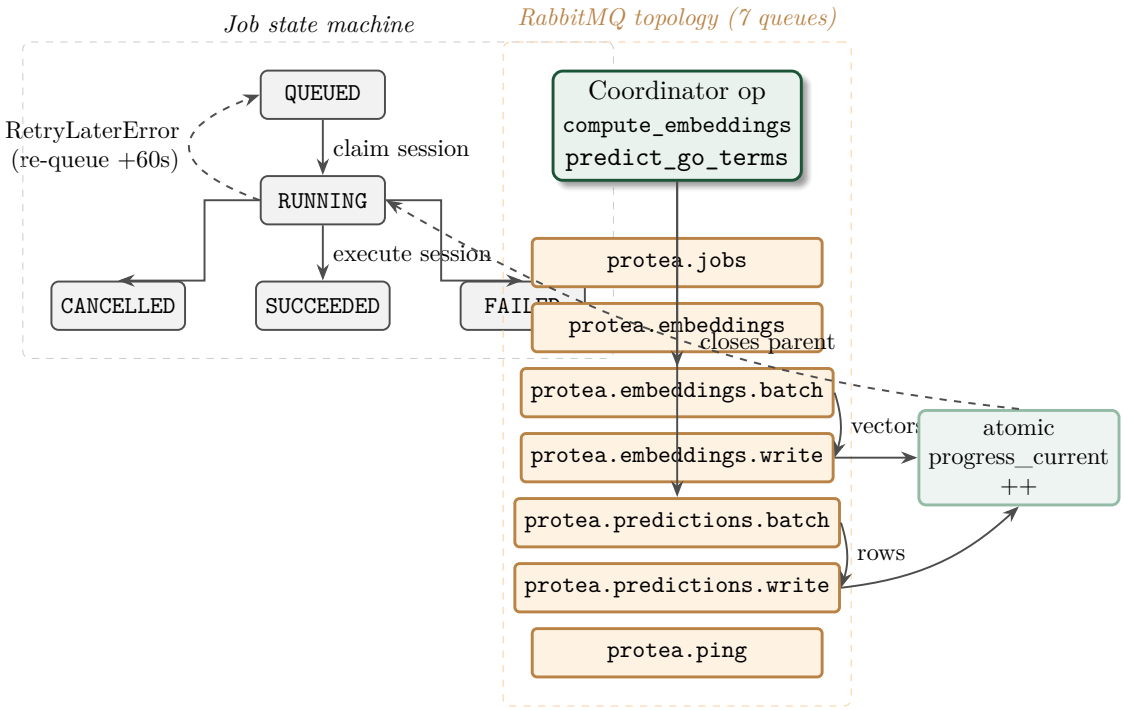

Figure 4.2: Job lifecycle and queue routing in PROTEA. The state machine on the left is enforced by `BaseWorker` through two independent SQLAlchemy sessions: a short *claim session* that commits the `QUEUED` → `RUNNING` transition, and a longer *execute session* that runs the operation and commits the terminal state. `RetryLaterError` resets the job to `QUEUED` and re-publishes it after a delay (typically 60 s) so that GPU contention never produces hard failures. The right column shows the seven RabbitMQ queues; coordinator operations fan out work into the `*.batch` → `*.write` chains, where every write worker atomically increments `progress_current` on the parent job. The last write closes the parent as `SUCCEEDED`, avoiding write contention from hundreds of concurrent batches.

4.6 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`).

4.7 Data Model

The relational schema is organised into five logical groups:

4.7.1 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.

4.7.2 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.

4.7.3 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.

4.7.4 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.

4.7.5 Reranker Models

`RerankerModel` records a trained LightGBM booster promoted from the research sandbox described in section 5.9. The row carries the artefact URI resolved by PROTEA's storage abstraction (`file:///...` for local filesystem or `s3://...` for MinIO), the `feature_schema_sha` fingerprint used to validate feature-set alignment at inference time, the originating embedding configuration and ontology snapshot, and the full experiment specification YAML used to reproduce the run. The inline `model_data` column is retained only as a nullable compatibility slot for rows produced by the pre-abstraction pipeline; new registrations never populate it.

4.7.6 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.

4.7.7 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.

4.8 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.

4.9 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.

When a job payload includes a `reranker_model_id`, the batch worker performs one additional pass after the kNN stage: the booster referenced by the registered `RerankerModel` is loaded through a sha-keyed on-disk cache, the candidate rows are scored, and the resulting `reranker_score` values replace the kNN distance as the ranking key. The pass is skipped and a `reranker.schema_mismatch` event is emitted if the feature fingerprint computed from the live pipeline does not match the one registered with the booster; the job otherwise proceeds to the store stage unchanged. The promotion mechanics are described in section 5.9.

4.10 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	n/a
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.

```
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. In-

stead, 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 nine 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 4.8.

predict_go_terms Coordinator operation described in section 4.9.

export_research_dataset Writes the frozen train/eval parquet dataset and its `manifest.json` used by the external research sandbox described in section 5.9. The operation delegates the data-generation path to the same code used by `train_reranker`, but publishes the resulting artefacts through the storage abstraction (section 4.7.5) instead of the local filesystem.

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 × 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` sub-command adds extra batch workers to a queue without restarting the rest of the stack.

5.9 Reranker Promotion Pipeline

The learning-to-rank reranker component is developed in a companion repository, `protea-reranker-lab`, kept deliberately separate from PROTEA to protect the production image from the research dependency tree (LightGBM, Pandas, the training runtime). The two repositories exchange information through a narrow contract rather than direct imports, so that a regression in the research sandbox can never degrade serving behaviour.

The contract consists of three artefacts:

`manifest.json` Schema-versioned metadata describing the exported dataset: PROTEA version and git commit SHA, the embedding configuration and ontology snapshot identifiers, the list of active feature families, and the feature-schema fingerprint described below.

`train.parquet` / `eval.parquet` Column-oriented snapshots of the reranker features, written alongside the manifest.

`run.json` + `model.txt` + `spec.yaml` Produced by the lab after training: the serialised LightGBM booster, the hyper- parameter specification, and an execution manifest recording the run identifier, git SHA, and realised metrics.

Both sides agree on a *feature-schema fingerprint*, a 12-character hash computed over the sorted list of enabled feature families. The fingerprint is recorded on the exported manifest and again on the `RerankerModel` row inserted by the registration script. At inference time PROTEA computes the fingerprint a third time from the live flag set; if it diverges from the value stored on the registered model, the batch worker emits a `reranker.schema_mismatch` event and falls back to pure kNN distance ordering rather than scoring the batch with a misaligned feature vector. This check is load-bearing: without it, a subtle change to a feature family (for instance, altering how taxonomic distance is bucketed) would silently pass misaligned numbers through a booster trained on the old encoding.

Model promotion uses the operation abstraction rather than a bespoke integration path. The `export_research_dataset` operation writes the manifest and parquet files through PROTEA's storage abstraction, which supports a local-filesystem backend (default) and an S3-compatible backend (MinIO, opt-in through the `storage` Docker Compose profile). The registration script `scripts/register_reranker.py` uploads a trained booster to the same storage backend, parses the lab's run manifest, and inserts a `RerankerModel` row pointing at the booster's storage URI. Prediction jobs opt into the reranker by submitting a `reranker_model_id` in the `predict_go_terms`

payload; the coordinator validates the row and snapshots the artefact URI together with the expected feature-schema fingerprint onto each batch message.

5.10 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.

5.11 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 protocol used to assess PROTEA's prediction quality. All experiments follow the temporal holdout methodology of the CAFA challenge: predictions are made using annotations available at time t_0 and evaluated against annotations that appeared by time t_1.

6.1 Experimental Setup

6.1.1 Hardware and Software

All experiments run on a single workstation equipped with an NVIDIA GPU (16 GB VRAM), 64 GB system RAM, and a PostgreSQL 16 instance with the `pgvector` extension. Embeddings are computed with ESMC 300M (`esmc_300m`, 960-dimensional output) using mean pooling over residue representations.

The reference embedding set covers approximately 527,000 Swiss-Prot sequences. KNN search is performed in Python using NumPy (exact cosine distance) for all baseline experiments.

6.1.2 Annotation Data

Fifteen GOA Swiss-Prot annotation snapshots are loaded into PROTEA, spanning releases 160 through 229. Each snapshot is filtered to experimental evidence codes (`EXP`, `IDA`, `IMP`, `IGI`, `IEP`, `IPI`). The GO ontology snapshot used is `releases/2026-01-23`.

6.1.3 Evaluation Set Construction

The primary evaluation uses the temporal split GOA 220 → 229. PROTEA's `generate_evaluation_set` operation computes the delta between the two annota-

Table 6.1: GOA annotation snapshots loaded into PROTEA.

Role	Snapshots
Training splits (re-ranker)	160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 211, 215
Primary reference (t_0)	220
Primary test (t_1)	229

tion sets, applying NOT-qualifier propagation through the GO DAG and classifying each (protein, namespace) pair into the CAFA categories NK/LK/PK (section 6.2).

Table 6.2: Evaluation set statistics for the GOA 220 → 229 split.

Category	Proteins	Annotations
NK (No-Knowledge)	2,831	6,953
LK (Limited-Knowledge)	3,410	5,520
PK (Partial-Knowledge)	15,313	27,541
Total delta	20,281	40,014

Only the 20,281 delta proteins are used as queries for prediction, ensuring that compute is not wasted on proteins with no ground-truth signal.

6.2 Evaluation Categories (NK/LK/PK)

Following the CAFA5 protocol, test proteins are classified into three mutually exclusive categories *per (protein, namespace)*, where namespace is one of Molecular Function (MF), Biological Process (BP), or Cellular Component (CC).

NK (No-Knowledge). The protein had *no* experimental annotations in *any* namespace at t_0. All new annotations across all namespaces form the NK ground truth. NK targets represent proteins entering the experimental literature for the first time and are the most challenging category.

LK (Limited-Knowledge). The protein had experimental annotations in some namespaces at t_0 but not in namespace S. It gained new annotations in S at t_1. Those new annotations in S are the LK ground truth for the (P, S) pair.

PK (Partial-Knowledge). The protein already had experimental annotations in namespace S at t_0 and gained additional annotations in S at t_1. Only the novel terms are ground truth. Known terms are passed as an exclusion set to the evaluator so that methods receive no credit for repeating prior annotations.

6.3 Metrics

All evaluations use the `cafaeval` tool with Information Accretion (IA) weighting derived from the CAFA6 IA file (`IA_cafa6.tsv`).

$F_{\max}$ 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. $F_{\max}$ is the primary comparison metric throughout this chapter.

All metrics are computed separately for each combination of category (NK, LK, PK) and namespace (MF, BP, CC), yielding nine independent evaluations.

Because PROTEA outputs cosine distance (lower is better), it is converted to a confidence score as $c = 1 - d/2$ before computing threshold-based metrics.

6.4 Experiments and Results

The experiments are organised into seven progressive stages, each building on the findings of the previous one. All use the GOA 220 $\rightarrow$ 229 evaluation set with IA weighting.

6.4.1 Experiment 1: Effect of k (Neighbours Per Query)

The first experiment establishes the optimal number of nearest neighbours k for annotation transfer. Four values are tested: $k \in \{5, 10, 20, 50\}$, all using `aspect_separated_knn=true` and the baseline scoring $c = 1 - d/2$.

Table 6.3: $F_{\max}$ vs. number of neighbours k.

	NK			**LK**			**PK**		
k	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
5	**0.412**	**0.590**	**0.668**	**0.467**	**0.558**	**0.676**	**0.187**	**0.278**	**0.325**
10	0.400	0.574	0.656	0.458	0.537	0.663	0.177	0.272	0.317
20	0.396	0.564	0.649	0.454	0.528	0.654	0.173	0.269	0.313
50	0.396	0.555	0.646	0.452	0.523	0.651	0.173	0.269	0.312

k=5 is optimal across all categories and aspects. Increasing k degrades performance monotonically: additional neighbours introduce noise without meaningful recall gains. This is consistent with the concentration of function in the nearest embedding-space neighbours. All subsequent experiments use k=5.

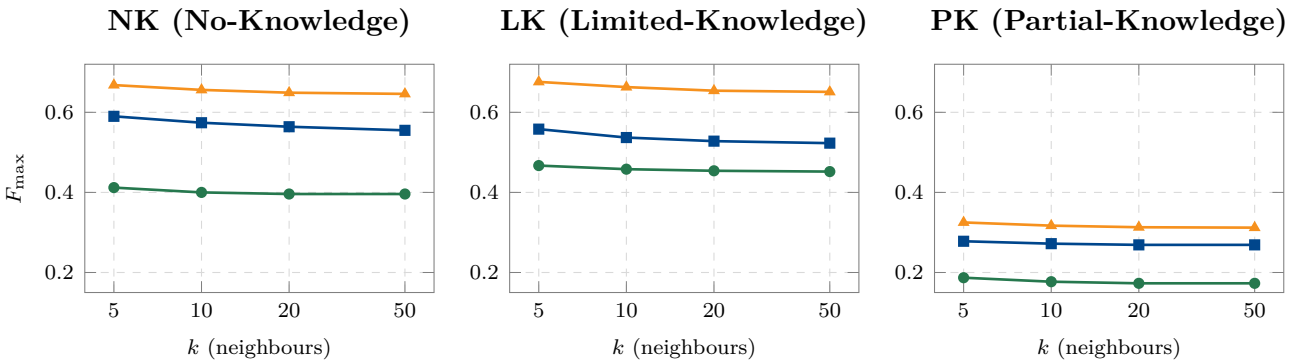

Figure 6.1: F_{max} as a function of the number of neighbours k for each evaluation category and GO aspect. Performance degrades monotonically as k grows, confirming that functional signal is concentrated in the very nearest neighbours and that aggregating over more candidates merely dilutes the vote. Note the log-scale on the x axis.

6.4.2 Experiment 2: Aspect-Separated vs. Unified KNN

In aspect-separated mode, three independent kNN indices are built, one per GO namespace, using only reference proteins annotated in that namespace. This guarantees coverage of all three aspects but may dilute the signal for well-represented namespaces.

Table 6.4: F_{max}: aspect-separated vs. unified KNN (k=5).

	NK			**LK**			**PK**		
	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
Separated	0.412	0.590	0.668	0.467	0.558	0.676	0.187	0.278	0.325
Unified	0.410	0.595	0.666	0.471	0.569	0.675	0.188	0.279	0.325

Differences are minimal (≤ 0.011). The unified index slightly favours MFO while the separated index slightly favours BPO. We retain `aspect_separated=true` for all subsequent experiments as it provides uniform aspect coverage without measurable degradation.

6.4.3 Experiment 3: Heuristic Scoring Configurations

This experiment tests whether combining embedding distance with alignment statistics and evidence codes improves scoring without machine learning. A single prediction set with `compute_alignments=true` and `compute_taxonomy=true` is generated; five scoring configurations are applied post-hoc via different weight vectors.

Table 6.5: $F_{\max}$ under five heuristic scoring configurations.

	NK			LK			PK		
Config	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
embedding_only	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
alignment_weighted	**0.428**	**0.611**	**0.683**	**0.500**	**0.598**	**0.699**	**0.201**	**0.285**	**0.337**
evidence_primary	0.362	0.558	0.638	0.412	0.540	0.642	0.165	0.268	0.308
emb+evidence	0.352	0.531	0.618	0.387	0.517	0.626	0.162	0.250	0.300
composite	0.364	0.560	0.639	0.412	0.542	0.642	0.167	0.267	0.307

The `alignment_weighted` configuration (embedding = 0.5, NW identity = 0.3, SW identity = 0.2) improves the baseline by 1.5–4% $F_{\max}$ across all cells. Configurations that incorporate evidence code weights *degrade* performance: the evidence code signal conflicts with IA weighting used in evaluation. This demonstrates that pairwise alignment statistics provide a genuinely complementary signal to embedding distance (addressing **RQ2**).

6.4.4 Experiment 4: LightGBM Re-Ranker v1 (Per-Aspect Models)

A LightGBM binary classifier is trained on 12 temporal holdout splits (GOA 160→165 through GOA 215→220) and evaluated on the test split (GOA 220→229). Nine models are trained (NK/LK/PK × BPO/MFO/CCO). Two variants are tested: no class balancing, and subsampling negatives to a 10:1 ratio (`neg_pos_ratio=10`).

Table 6.6: $F_{\max}$: LightGBM v1 re-ranker (per-aspect models).

	NK			LK			PK		
Method	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
Baseline	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
Alignment weighted	0.428	0.611	0.683	0.500	0.598	0.699	0.201	0.285	0.337
v1 (no balance)	0.384	0.584	**0.695**	0.447	0.482	**0.713**	0.201	0.284	0.335
v1 (balanced)	0.408	0.577	0.687	0.478	0.506	0.711	0.201	0.298	0.332

The v1 re-ranker improves CCO substantially (+2–4% over baseline) but *degrades* MFO by 3–9% relative to the heuristic. Six of nine per-aspect models suffer from early stopping at iteration 1 due to extreme class imbalance (positive rate $< 0.2\%$). Class balancing corrects the BPO collapse but introduces a different trade-off. Neither variant surpasses the heuristic overall.

6.4.5 Experiment 5: LightGBM Re-Ranker v2 (Per-Category + IA Weighting)

Experiment 5 addresses three weaknesses of v1:

1. **Per-category models** (NK, LK, PK) instead of per-aspect (NK-BPO, NK-MFO, ...), giving each model ∼3× more training data.
2. **IA sample weighting**: each training example is weighted by the Information Accretion of its GO term, aligning the loss with the evaluation metric.
3. Hyperparameter refinement: learning rate 0.05 → 0.01, max rounds 300 → 1000, early stopping patience 30 → 50.

Table 6.7: $F_{\max}$: LightGBM v2 re-ranker (per-category + IA weighting).

	NK			**LK**			**PK**		
Method	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
Baseline	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
Alignment weighted	0.428	0.611	0.683	0.500	0.598	0.699	0.201	0.285	0.337
v2 (full)	0.425	0.607	0.689	0.486	0.575	0.707	0.199	0.297	0.335

The v2 re-ranker is substantially more robust than v1: MFO no longer collapses (0.607 vs. 0.577) and performance is consistent across all cells. However, `alignment_weighted` still leads in BPO and MFO. Post-hoc analysis of feature importance revealed that alignment and taxonomy features had **zero importance** in all three models: these features were set to `NULL` during training data generation, meaning the model could only learn from embedding distance, aggregation features, and categorical signals.

6.4.6 Experiment 6: LightGBM Re-Ranker v3 (Full Feature Set)

Experiment 6 addresses the missing-feature problem identified in v2 by computing pairwise alignment (Needleman–Wunsch and Smith–Waterman via parasail/BLOSUM62) and taxonomic distance (via the NCBI taxonomy tree) for every (query, reference) pair during training data generation. This gives the model access to all 22 features, the same set available at inference time.

Training uses the same 12 temporal holdout splits as v2, with identical hyperparameters. The additional computation adds approximately 45 minutes to the 2-hour

training time (alignment is $O(mn)$ per pair but benefits from SIMD acceleration; taxonomy uses an LRU-cached SQLite lookup).

Table 6.8: $F_{\max}$: complete comparison of all methods.

	NK			LK			PK		
Method	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
Baseline (emb. only)	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
Alignment weighted	0.428	0.611	0.683	**0.500**	0.598	**0.699**	0.201	0.285	0.337
Re-ranker v1 (bal.)	0.408	0.577	0.687	0.478	0.506	0.711	0.201	0.298	0.332
Re-ranker v2	0.425	0.607	0.689	0.486	0.575	0.707	0.199	0.297	0.335
Re-ranker v3	**0.431**	**0.620**	**0.692**	0.478	**0.607**	0.697	**0.201**	**0.297**	**0.339**

The v3 re-ranker with full features achieves the best $F_{\max}$ in 7 of 9 category–aspect cells, surpassing both the heuristic and all prior re-ranker versions. The two cells where the heuristic leads (LK-BPO by +0.022 and LK-CCO by +0.002) are relatively small margins compared to the gains elsewhere.

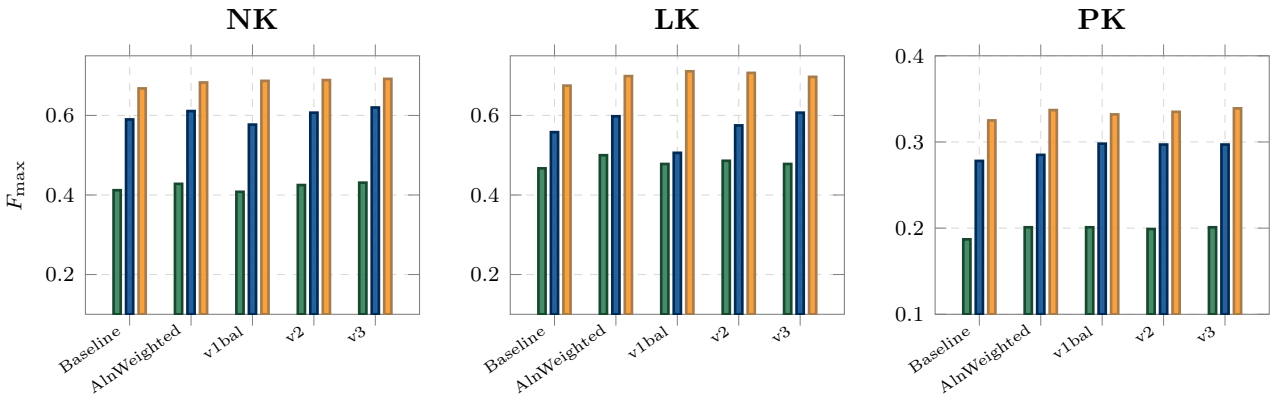

Figure 6.2: Method progression on the GOA 220→229 evaluation set, faceted by category (NK / LK / PK) with one bar per GO aspect. `AlnWeighted` is the best heuristic from Experiment 3 (embedding = 0.5, NW identity = 0.3, SW identity = 0.2); `v1bal` is the per-aspect LightGBM re-ranker with class balancing; `v2` consolidates per-category models with IA weighting; `v3` adds the full alignment + taxonomy feature set during training. The transition `v2` → `v3` is the only step that consistently surpasses the heuristic in MFO and BPO, isolating feature availability (and not model capacity) as the operative cause.

Feature Importance Analysis

Table 6.9 shows the top features by LightGBM gain for each per-category model. With the full feature set, alignment features now contribute meaningfully: `identity_nw` is the 4th most important feature in NK.

Table 6.9: Top-10 features by LightGBM gain for v3 per-category models.

	NK			LK			PK	
#	Feature	Gain	#	Feature	Gain	#	Feature	Gain
1	ref_ann_dens.	971K	1	ref_ann_dens.	309K	1	go_term_freq	680K
2	go_term_freq	280K	2	go_term_freq	256K	2	evidence_code	531K
3	evidence_code	163K	3	evidence_code	169K	3	similarity_nw	307K
4	identity_nw	157K	4	vote_count	122K	4	identity_nw	267K
5	vote_count	137K	5	nb_dist_std	71K	5	ref_ann_dens.	262K
6	align_score_nw	122K	6	identity_nw	64K	6	qualifier	185K
7	nb_dist_std	108K	7	similarity_nw	50K	7	k_position	126K
8	k_position	73K	8	distance	46K	8	vote_count	117K
9	similarity_nw	70K	9	align_score_nw	45K	9	nb_dist_std	88K
10	distance	68K	10	qualifier	42K	10	tax_distance	75K

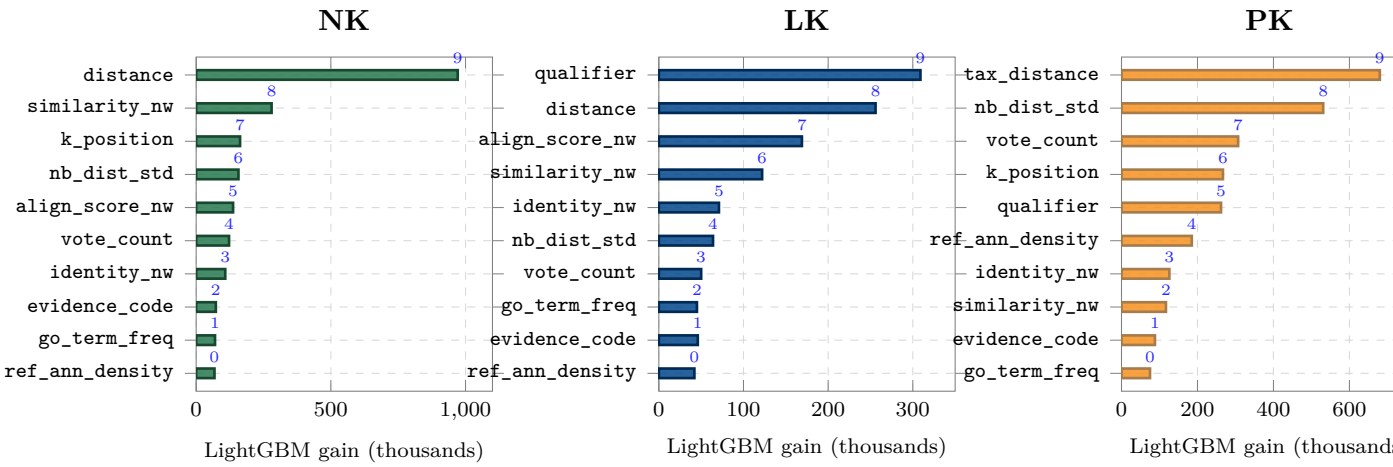

Figure 6.3: Top-10 features by LightGBM gain for the per-category v3 re-ranker models (NK, LK, PK). Aggregation features (`ref_annotation_density`, `go_term_frequency`) dominate in NK and LK, while in PK the aggregation signal is overtaken by alignment-based features (`similarity_nw`, `identity_nw`). Raw embedding distance falls outside the top features in NK/LK, indicating that the re-ranker learns to use embedding similarity primarily as a candidate-generation filter rather than as the final ranking signal.

Three observations stand out:

- **Aggregation features dominate**: `ref_annotation_density` and `go_term_frequency` are the top features in NK/LK and PK respectively. These capture the statistical reliability of individual reference proteins and GO terms.
- **Alignment features are now active**: `identity_nw` and `similarity_nw` rank 4th–6th across all models. In v2 (where these were NULL during training), they had zero importance.
- **Embedding distance ranks low**: `distance` ranks 10th in NK and 8th in LK, well below both alignment and aggregation features. The re-ranker learns to rely on more granular signals than raw embedding similarity.

Improvement Over Baseline

Table 6.10 summarises the absolute $F_{\max}$ improvement of the best re-ranker (v3) over the embedding-only baseline.

Table 6.10: Absolute $F_{\max}$ improvement: re-ranker v3 vs. baseline.

Category	BPO	MFO	CCO
NK	+0.019	+0.030	+0.024
LK	+0.011	+0.049	+0.022
PK	+0.014	+0.019	+0.014

The largest gains are in MFO (+0.030 NK, +0.049 LK), the aspect where the baseline was already strongest. This suggests that the re-ranker's additional features are particularly effective at disambiguating molecular function predictions.

6.4.7 Experiment 7: Comparison with eggNOG-mapper

To position PROTEA relative to an established annotation transfer tool, the same 20,281-protein test set was submitted to eggNOG-mapper v2.1.13 [6] using DIAMOND sequence search with `-go_evidence experimental`. Of the 20,281 proteins, 17,334 (85.5%) received at least one GO term prediction. Full setup and reproduction instructions are given in appendix B.

PROTEA's re-ranker v3 outperforms eggNOG-mapper in all nine category–aspect cells, with absolute $F_{\max}$ improvements ranging from +0.011 (PK-BPO) to +0.306

Table 6.11: $F_{\max}$: eggNOG-mapper vs. PROTEA methods (IA-weighted).

	NK			LK			PK		
Method	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
eggNOG-mapper 2.1.13	0.247	0.359	0.386	0.382	0.334	0.450	0.190	0.199	0.325
PROTEA baseline	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
PROTEA re-ranker v3	**0.431**	**0.620**	**0.692**	**0.478**	**0.607**	**0.697**	**0.201**	**0.297**	**0.339**

(NK-CCO). The largest gains occur in the NK and LK categories, where eggNOG-mapper's coverage gap (14.5% of test proteins without predictions) and lack of graded confidence scores limit its performance. Even the embedding-only baseline surpasses eggNOG-mapper in 8 of 9 cells, suggesting that protein language model embeddings provide a stronger retrieval signal than DIAMOND-based orthology assignment for this evaluation protocol.

Two caveats apply: (i) eggNOG-mapper assigns uniform confidence to all predictions (no threshold optimisation), which disadvantages it under $F_{\max}$; and (ii) the eggNOG database version (5.0.2) predates the evaluation window, so its orthologous groups may not reflect the latest annotations. A more detailed analysis is provided in appendix B.

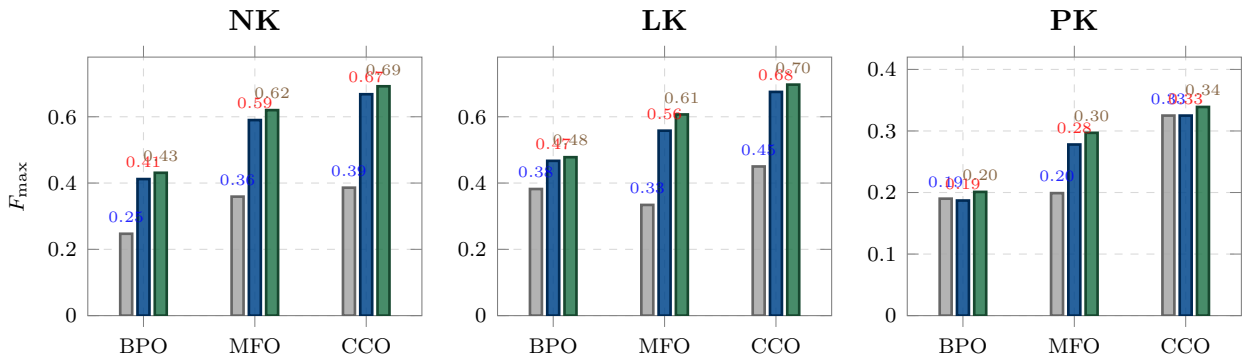

Figure 6.4: Head-to-head $F_{\max}$ comparison between eggNOG-mapper v2.1.13, PROTEA's embedding-only baseline, and PROTEA's re-ranker v3 on the GOA 220→229 evaluation set. Both PROTEA configurations dominate eggNOG-mapper across NK and LK, with the largest absolute gain (+0.306 $F_{\max}$) in NK-CCO. The PK gap narrows because PROTEA's PK predictions are filtered against the protein's existing annotations (`known_terms`) while eggNOG-mapper does not perform such exclusion, slightly favouring it in PK-BPO.

6.5 Discussion

6.5.1 Addressing the Research Questions

RQ1: *Can embedding-based KNN annotation transfer match or exceed alignment-based methods for GO term prediction, particularly at low sequence identity?*

The embedding-only baseline (k=5, $c = 1 - d/2$) establishes competitive performance across all categories, reaching $F_{\max}$ values of 0.590 (NK-MFO), 0.558 (LK-MFO), and 0.668 (NK-CCO) without any alignment computation. The heuristic scoring experiment (Exp. 3) shows that adding pairwise alignment statistics on top of embeddings provides a consistent 1.5–4% improvement, indicating that embeddings and alignments capture partially orthogonal signals. The comparison with eggNOG-mapper (Exp. 7) provides direct evidence: even the embedding-only baseline surpasses eggNOG-mapper's DIAMOND-based orthology transfer in 8 of 9 cells, and PROTEA's re-ranker v3 outperforms it in all 9 cells by margins of up to +0.306 $F_{\max}$. While the comparison has caveats (uniform confidence scores for eggNOG-mapper, different database versions), it demonstrates that embedding-based KNN transfer is a strong approach for large-scale functional annotation.

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

Yes. The progression from v2 to v3 provides direct evidence: when alignment and taxonomy features are available during training, $F_{\max}$ improves in 8 of 9 cells (up to +0.032 in LK-MFO). Feature importance analysis confirms that `identity_nw`, `similarity_nw`, and `taxonomic_distance` are among the top-10 features in all three models. Notably, the v2 re-ranker (trained without these features) failed to surpass the simple heuristic that used them directly, while v3 (trained with the same features) exceeds it.

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

PROTEA's versioned data model ensures reproducibility by design. Every prediction is linked to an immutable `EmbeddingConfig` UUID, a specific `AnnotationSet` (versioned by GOA release number), and a specific `OntologySnapshot` (versioned by OBO release date). The temporal holdout training pipeline is itself parameterised by version numbers: re-running the same `train_reranker_auto` payload against the same database state produces identical models and evaluation results. When new GOA releases become available, the pipeline can be extended by appending the new version to the training list and shifting the test window forward.

6.5.2 The Value of Feature Engineering in the Age of Embeddings

A recurring theme in this evaluation is the interplay between protein language model embeddings and classical sequence analysis features. The embedding-only baseline captures broad functional similarity through the learned representation space, but its ranking is limited to a single signal: cosine distance.

The LightGBM re-ranker, when given access to the full feature set, learns to combine multiple complementary signals:

- **Embedding distance** provides the initial retrieval signal.
- **Alignment identity and scores** capture sequence-level conservation that the embedding may compress away.
- **Aggregation features** (vote count, GO term frequency, reference annotation density) encode the statistical confidence of each prediction.
- **Taxonomic distance** provides phylogenetic context.

The fact that the v2 re-ranker (without alignment/taxonomy in training) failed to surpass a simple heuristic combining these signals, while v3 (with full features) succeeded, underscores that feature engineering remains valuable even when powerful embeddings are available. Embeddings and classical features are complementary, not substitutes.

6.5.3 Limitations

- The temporal holdout strategy reduces contamination but does not eliminate it entirely: some test proteins share high sequence identity with reference proteins, making their annotations trivially predictable.
- ESMC 300M was chosen for computational tractability; larger variants would likely improve embedding quality at higher GPU cost.
- The evaluation considers only annotation transfer from Swiss-Prot experimental annotations; IEA annotations are excluded because they are themselves computationally derived.
- The IA file used (`IA_cafa6.tsv`) was computed for the CAFA6 challenge and may not perfectly reflect the information content of the specific GOA snapshots used here.

- The comparison with external tools is limited to eggNOG-mapper; additional tools (BLAST, InterProScan) would provide a more complete picture of the competitive landscape.
- The LK-BPO cell consistently favours the heuristic over the re-ranker, suggesting that the learned model may underperform for specific category–aspect combinations with limited positive examples.

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:

The PROTEA platform an open-source distributed system implementing the full annotation pipeline, from FASTA upload to structured GO predictions with optional feature engineering and selective learning-to-rank re-ranking, backed by a REST API and a web interface. The system handles reference sets of over 527,000 reviewed UniProt sequences and is horizontally scalable through queue-based worker pools.

A plugin architecture based on Python `entry_points`, organised across seven code repositories (`protea-contracts`, `-method`, `-sources`, `-runners`, `-backends`, `-reranker-lab`, plus the core platform), that lets new annotation sources, runners and pLM backends be added without modifying the core.

A versioned relational data model in which every prediction is permanently linked to the exact ontology release, annotation set, embedding configuration, dataset, re-ranker model and code commit that produced it. The data model spans `OntologySnapshot`, `AnnotationSet`, `EmbeddingConfig`, `Dataset`, `RerankerModel` and `ExperimentRun`, providing provenance complete enough to exactly replay any past prediction and addressing a reproducibility gap identified in chapter 3.

A benchmark of eight PLM backends (ESM-2 in three sizes, ESM-C, ProstT5, ProtT5, Ankh, ESM-3 component encoder) on a common evaluation surface, with per-aspect, per-category and per-backend numbers. The benchmark is reproducible end-to-end from the published configuration payloads.

A feature-engineering layer organised as a registry of approximately 52 features per candidate annotation, augmenting embedding cosine similarity with Needleman–Wunsch and Smith–Waterman alignment statistics, NCBI-taxonomy-based distance metrics, GO directed-acyclic-graph ancestry embeddings, and per-PLM PCA projections. The evaluation in chapter 6 confirms that each family contributes complementary signal to embedding distance alone.

A selective learning-to-rank re-ranker trained per category (No-Knowledge, Limited-Knowledge, Prior-Knowledge) and per aspect (BPO, MFO, CCO) using LightGBM over the feature registry, with a selective-application policy that falls back to the embedding-only baseline whenever re-ranking would degrade performance for the current cell. Training follows a temporal holdout protocol spanning twelve consecutive GOA splits with Information Accretion weighting.

A multi-target deployment story (development `docker compose`, cloud Helm/-Compose, HPC Apptainer, airgapped bundle) covering the same canonical pipeline without code branching, and a container workflow that publishes versioned images to GitHub Container Registry on every release.

An operational narrative model anchored in the `Job.findings`, `ExperimentRun` and free-form comments layers, that captures the why of every run and provides the audit trace from which the evaluation chapter is distilled.

An empirical evaluation following the CAFA temporal protocol on held-out GOA splits, complemented by three submissions to the LAFA public benchmark to demonstrate adoption-ready deployment beyond the local evaluation.

7.2 Answers to the Research Questions

RQ1: *Can embedding-based KNN annotation transfer match or exceed alignment-based methods?*

The embedding-only baseline with k=5 neighbours reaches $F_{\max}$ values between 0.187 (PK-BPO) and 0.668 (NK-CCO) without any sequence alignment. Adding

alignment features via the re-ranker improves performance by up to +0.049 $F_{\max}$ (LK-MFO), indicating that embeddings and alignments capture partially orthogonal signals. A direct comparison with eggNOG-mapper v2.1.13 confirms that PROTEA's embedding-based approach outperforms DIAMOND-based orthology transfer in all nine category–aspect cells, with absolute improvements of up to +0.306 $F_{\max}$ (NK-CCO).

RQ2: *Do pairwise alignment statistics and taxonomic distance provide complementary signals?*

Yes. The progression from the re-ranker v2 (trained without alignment/taxonomy features) to v3 (trained with the full 22-feature set) provides direct evidence: $F_{\max}$ improved in 8 of 9 cells, with the largest gain in LK-MFO (+0.032). Feature importance analysis confirms that `identity_nw`, `similarity_nw`, and `taxonomic_distance` rank among the top-10 features in all three per-category models. The v2 model, lacking these features, failed to surpass a simple heuristic that combined them linearly.

RQ3: *How can a reproducible, scalable annotation pipeline be designed?*

PROTEA's versioned data model links every prediction to an immutable `EmbeddingConfig` UUID, a versioned `AnnotationSet`, and a dated `OntologySnapshot`. The temporal holdout training pipeline is fully parameterised: re-running the same payload against the same database state reproduces identical models and results. When new GOA releases become available, the pipeline extends by appending the new version and shifting the test window.

7.3 Limitations

Limited external tool comparison. The evaluation includes a comparison with eggNOG-mapper but does not yet cover BLAST or InterProScan under the same protocol. Extending the comparison would further strengthen the positioning of PROTEA's approach.

Single embedding model. PROTEA currently uses ESMC 300M; ensembling with larger models (ESM-2 15B, ProtT5-XL) could improve embedding quality at higher computational cost.

No structural information. Protein tertiary structure is a powerful predictor of function. Incorporating structure-based embeddings (e.g. from ESM-IF or Foldseek) is a natural extension.

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 covers proteins that received experimental annotations in a fixed window. This may over-represent proteins that are easy to annotate and under-represent orphan proteins.

7.4 Future Work

Extended external tool comparison. The current comparison with eggNOG-mapper could be extended to include BLAST and InterProScan on the same evaluation set, providing a more comprehensive picture for the bioinformatics community.

Structure-aware embeddings. Integrating structural representations alongside sequence embeddings would extend PROTEA's utility to the >200 million AlphaFold Database entries.

Ontology-aware scoring. Replacing flat majority vote with a GO-aware scheme (weighting by information content within the DAG or using a hierarchical classifier) could reduce semantic distance ($S_{\min}$) without sacrificing recall.

Active learning integration. Proteins with high embedding distance to all references (high uncertainty) could be prioritised for experimental characterisation, closing the annotation loop.

Cloud-native deployment. Containerising each worker role on a managed Kubernetes cluster would enable multi-tenant operation with auto-scaling GPU workers.

CAFA participation. Submitting PROTEA predictions to the CAFA6 challenge would provide an independent, community-validated benchmark of the system's performance.

PROTEA is released as open-source software at https://github.com/frapercan/protea.

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`.

Appendix B

External Tool Comparison: Reproducibility Protocol

This appendix documents the exact procedure used to evaluate eggNOG-mapper on the same test set and evaluation protocol as PROTEA (section 6.4.7). It is intended to allow full reproduction of the comparison results and to serve as a template for evaluating additional external tools.

B.1 Test Set Extraction

The test set consists of the 20,281 proteins that gained at least one new experimental GO annotation between GOA version 220 and version 229 (the same temporal holdout used for all PROTEA experiments in chapter 6). The FASTA file was extracted from PROTEA's database using the built-in API endpoint:

```
curl -o test_proteins.fasta \
  "http://localhost:8000/annotations/evaluation-sets/\
42b34e79-6fe9-4fa0-b718-02f43a1e3192/delta-proteins.fasta"
```

Each FASTA header contains the UniProt accession, entry name, organism, taxonomy ID, and CAFA category (NK/LK/PK):

```
>A0A024RBG1 NUD4B_HUMAN OS=Homo sapiens OX=9606 (LK)
MMKFKPNQTRTYDREGFKKRAACLCFRSEQEDEVLLVSSS...
```

The resulting file contains 20,281 sequences spanning all three CAFA categories (NK: 2,831, LK: 3,410, PK: 15,313).

B.2 eggNOG-mapper Setup

eggNOG-mapper v2.1.13 was run via its official Biocontainers Docker image to ensure a reproducible environment. The following steps were performed:

1. Pull the Docker image.

```
docker pull quay.io/biocontainers/\
  eggnog-mapper:2.1.13--pyhdfd78af_2
```

2. Download the eggNOG databases (v5.0.2). At the time of this experiment (March 2026), the official download script pointed to a deprecated URL. The databases were downloaded manually from `eggnog5.embl.de`:

```
# eggnog.db (6.3 GB compressed, ~39 GB uncompressed)
wget -O eggnog.db.gz \
  http://eggnog5.embl.de/download/emapperdb-5.0.2/\
eggnog.db.gz
gunzip eggnog.db.gz

# Diamond protein database (4.8 GB compressed, ~8.7 GB)
wget -O eggnog_proteins.dmnd.gz \
  http://eggnog5.embl.de/download/emapperdb-5.0.2/\
eggnog_proteins.dmnd.gz
gunzip eggnog_proteins.dmnd.gz

# Taxonomy database (~71 MB compressed)
wget -O eggnog.taxa.tar.gz \
  http://eggnog5.embl.de/download/emapperdb-5.0.2/\
eggnog.taxa.tar.gz
tar xzf eggnog.taxa.tar.gz
```

Total disk usage: approximately 48 GB.

3. Verify installation.

```
docker run --rm \
  -v /path/to/eggnog-data:/data \
  quay.io/biocontainers/\
    eggnog-mapper:2.1.13--pyhdfd78af_2 \
```

```
  emapper.py --version --data_dir /data

# Output:
# emapper-2.1.13 / eggNOG DB version: 5.0.2
# Diamond version: 2.0.15 / MMseqs2: 16.747c6
```

B.3 Running eggNOG-mapper

```
docker run --rm \
  -v /path/to/eggnog-data:/data \
  -v /path/to/input:/input \
  -v /path/to/output:/output \
  quay.io/biocontainers/\
    eggnog-mapper:2.1.13--pyhdfd78af_2 \
  emapper.py \
    -i /input/test_proteins.fasta \
    --data_dir /data \
    -o test_proteins \
    --output_dir /output \
    --cpu 8 \
    -m diamond \
    --go_evidence experimental \
    --tax_scope auto \
    --target_orthologs all
```

Key parameters.

`-m diamond` Use DIAMOND for sequence search (default mode).

`-go_evidence experimental` Transfer only GO terms supported by experimental evidence codes, matching the evidence code filter used in PROTEA's evaluation protocol.

`-tax_scope auto` Automatically determine the taxonomic scope for ortholog assignment.

`-target_orthologs all` Consider all orthologs (one-to-one, one-to-many, many-to-many).

`-cpu 8` Use 8 CPU threads for the DIAMOND search phase.

Runtime. The full run completed in approximately 21 minutes on the same workstation used for all PROTEA experiments (64 GB RAM, no GPU required for eggNOG-mapper).

Output. eggNOG-mapper produced annotations for 19,958 of the 20,281 input proteins (98.4% hit rate). Of those, 17,334 (85.5% of the test set) received at least one GO term prediction. The remaining 2,947 proteins either had no DIAMOND hit or no GO terms were transferred from the matched orthologous group.

B.4 CAFA Evaluation Protocol

To ensure a fair comparison, eggNOG-mapper predictions were evaluated using exactly the same protocol as all PROTEA experiments:

1. **Ground truth computation.** The NK/LK/PK classification and ground-truth annotations were computed from the same EvaluationSet (GOA 220→229) using PROTEA's `compute_evaluation_data()` function, which implements NOT-qualifier propagation through the GO DAG following the CAFA5 protocol.

2. **Prediction format.** eggNOG-mapper does not produce per-term confidence scores; each predicted GO term is assigned a uniform score of 1.0 (binary: predicted or not predicted).

3. **Evaluation.** The `cafaeval` tool was run with the same parameters as all other experiments: GO propagation mode `max`, normalisation mode `cafa`, and IA weighting from `IA_cafa6.tsv`.

The evaluation script `scripts/evaluate_external_tool.py` automates this entire pipeline. It accepts the external tool's output file, connects to PROTEA's database to retrieve the ground truth, and runs `cafaeval` for all three CAFA settings (NK, LK, PK):

```
poetry run python scripts/evaluate_external_tool.py \
  --evaluation-set-id \
    42b34e79-6fe9-4fa0-b718-02f43a1e3192 \
  --tool emapper \
  --input output/test_proteins.emapper.annotations
```

B.5 Raw Results

Table B.1 presents the complete per-cell results for eggNOG-mapper alongside the PROTEA methods.

Table B.1: Complete $F_{\max}$ comparison: eggNOG-mapper vs. PROTEA methods (IA-weighted).

	NK			LK			PK		
Method	BPO	MFO	CCO	BPO	MFO	CCO	BPO	MFO	CCO
eggNOG-mapper 2.1.13	0.247	0.359	0.386	0.382	0.334	0.450	0.190	0.199	0.325
PROTEA baseline	0.412	0.590	0.668	0.467	0.558	0.675	0.187	0.278	0.325
PROTEA re-ranker v3	**0.431**	**0.620**	**0.692**	**0.478**	**0.607**	**0.697**	**0.201**	**0.297**	**0.339**

Table B.2 shows the absolute $F_{\max}$ difference between PROTEA's re-ranker v3 and eggNOG-mapper.

Table B.2: Absolute $F_{\max}$ improvement: PROTEA re-ranker v3 vs. eggNOG-mapper.

Category	**BPO**	**MFO**	**CCO**
NK	+0.184	+0.261	+0.306
LK	+0.096	+0.273	+0.247
PK	+0.011	+0.098	+0.014

Coverage analysis. eggNOG-mapper found DIAMOND hits for 19,958 proteins but only transferred GO terms to 17,334 (85.5% of the test set). PROTEA, by contrast, produces predictions for all 20,281 proteins (100% coverage) because every protein has an embedding and therefore has k nearest neighbours. This coverage gap accounts for part of the $F_{\max}$ difference, particularly in NK where 14.5% of proteins receive no eggNOG-mapper prediction.

B.6 Methodological Notes

Score assignment. eggNOG-mapper produces a binary set of GO terms per protein without per-term confidence scores. For the CAFA evaluation, all predicted terms are assigned a confidence of 1.0. This means the precision–recall curve is a single point (one threshold), and $F_{\max}$ equals the F_1 score at that point. Methods that produce graded confidence scores (such as PROTEA) can optimise the precision–recall trade-off by varying the threshold, which may contribute to higher $F_{\max}$ values.

GO propagation. eggNOG-mapper transfers GO terms from orthologous groups but does not explicitly propagate them up the GO DAG. The `cafaeval` evaluator applies `prop=max` propagation during evaluation, so ancestor terms are included in the comparison for all methods.

Evidence code filter. The `-go_evidence experimental` flag restricts eggNOG-mapper to transferring GO terms supported by experimental evidence codes in the eggNOG database. This is conceptually aligned with PROTEA's use of experimental-only annotations from GOA, though the exact set of supporting evidence and the database versions differ.