



MDMC

Master in Data Management
and Curation

MASTER IN DATA MANAGEMENT AND
CURATION

Web Application for DPCfam and
DPCstruct Data Exploration

Supervisor(s):

Dr. Valerio PIOMPONI,

Dr. Elaheh SAADAT

Candidate:

Emmanuel NYANDU KAGARABI

2025 - 2026



This course has been funded by the European Union – NextGenerationEU within the projects:

- *NFFA-DI cod. IR00000015 - Missione 4, “Istruzione e Ricerca” – Componente 2, “Dalla ricerca all’impresa” – Linea di investimento 3.1, “Fondo per la realizzazione di un sistema integrato di infrastrutture di ricerca e innovazione” – Azione 3.1.1, “Creazione di nuove IR o potenziamento di quelle esistenti che concorrono agli obiettivi di Eccellenza Scientifica di Horizon Europe e costituzione di reti” (CUP B53C22004310006)*

[NFFA-DI : Nano Foundries Fine Analysis - Digital Infrastructure](#)

- *EFC cod. SSU2024 – 00002 - Missione 4, “Istruzione e Ricerca” – Componente 1, “Potenziamento dell’offerta dei servizi all’istruzione: dagli asili nido all’ università” – Linea di investimento 3.4, “Didattica e competenze universitarie avanzate” – Sub-Investimento, “Rafforzamento delle scuole universitarie superiori” (CUP G97G24000100007)*

[EFC : Educating Future Citizen](#)



Author's Declaration

I, **Emmanuel Nyandu Kagarabi**, declare that this thesis entitled, “**Web Application for DPCfam and DPCstruct Data Exploration**” and the work presented therein are my own. I certify that:

- This work was performed wholly or principally during MDMC internship in the **Laboratory of Data Engineering** (LADE/Area Science Park).
- If any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have relied on the published work of others, this is always clearly attributed.
- Where the thesis is based on work I have done in collaboration with others, I have made clear exactly what was done by others and what I contributed.
- With respect to AI technologies:
 - I acknowledge the use of **Claude by Anthropic** (<https://claude.ai>) to provide information for background research and to assist in the drafting writing process with the creation of an outline structure for this essay.
 - I acknowledge the use of **Gemini** (<https://gemini.google.com>) as a source of information to create materials that will be used in my own words.
 - I acknowledge the use of **GitHub Copilot** (<https://github.com/features/copilot>) within Visual Studio Code as an AI programming assistant to help speed up development and improve my coding skills.
 - No content generated by AI technologies was presented as my own work.
- Where I have quoted from the work of others, the source is always cited. Except for such quotations, this thesis is entirely my own work.
- I have acknowledged all major sources of help.



June 30, 2026

“The man who moves a mountain begins by carrying away small stones.”

— **Confucius**

“What I can not create, I do not understand.”

— **Richard Feynman**

“All models are wrong, but some are useful.”

— **George Edward Pelham Box**

Acknowledgments

First of all, I thank God, the Master of time and circumstances, who makes a way where there seems to be no way and at times when all hope appears lost.

I sincerely thank **SISSA** and **Area Science Park** for granting me a place in this exceptional Master's programme and for making it possible through a fully-funded scholarship. The road was long, but you never let me go: from the stressful visa process back in Kinshasa and Brazzaville, through the residence permit marathon, all the way to the end, and beyond. In particular, I thank the **ILAS Office** (SABRINA & CLAUDIA), the **Mobility Office** (GABRIELLA), and the MDMC programme coordination (**Prof. Eugenio & Dr. Mariarita**). I simply cannot find the right words: *Grazie mille!*

In a very special way, I thank my supervisors **Dr. Valerio Piomponi** and **Dr. Elaheh Saadat** for being the best coaches one could hope for. Your guidance, patience, and generosity of time made this work possible. I am deeply grateful to you both.

To my friends and colleagues in Trieste: Nando, Ruth, Lesly, Luis-Fernando, Valentin, Dariose, and the others; thank you for the warmth and support along the way. A very special thank you to **Richard & Ruth**, who welcomed me into their home with a generosity I will never forget.

To my family and friends back home in the DRC: you are too many to name, but you know who you are. You supported me through everything, including this dark period where the M23 war continues to ravage our city. From the bottom of my heart: *Merci infiniment à vous!*

Finally, I thank **Area Science Park** through the entire staff of the Laboratory of Data Engineering (LADE) - RIT for the productive and welcoming environment. A special acknowledgment goes to **Federico Barone**, whose PhD thesis gave birth to this project and whose initial *dpcfam web application* was our starting point. I also thank the **IT team** for their precious assistance during the development and deployment of our application on the ORFEO HPC cluster.

Abstract

The rapid growth of protein sequence and structure databases has created a significant **annotation gap** that manual curation alone cannot bridge. Two recent computational biology methods, **DPCfam** and **DPCstruct**, address this by applying the **Density Peak Clustering (DPC)** algorithm to automatically group millions of protein domains into putative evolutionary families called **metaclusters**, based purely on pairwise similarity scores derived from large-scale sequence or structural alignments. This process has yielded **81,384** sequence-based and **28,246** structure-based families with no manual curation. Although the datasets derived from both methods are publicly available on Zenodo under FAIR data principles, exploring them typically requires downloading and parsing gigabyte-scale archives, including a massive **81 GB XML file** for DPCfam, which presents a technical barrier for many researchers.

In this thesis, we present **DPCexplorer**, a unified **Django web application** that closes this gap. Raw archives were preprocessed on the **ORFEO HPC cluster** and loaded into a **PostgreSQL** database optimized with B-Tree, GIN trigram, and functional indexes. Built on the Model-View-Template (MVT) pattern, the platform accepts four query types (**DPCfam MCID**, **DPCstruct MCID**, **Pfam ID**, and **UniProt accession**) and returns paginated metadata tables, per-protein domain-architecture diagrams, and downloadable biological files (FASTA, MSA, HMM, PDB). Furthermore, for DPCstruct metaclusters, it integrates the **PDBe-Mol*** 3D viewer to render AlphaFold2-predicted structures colored by per-residue pLDDT confidence directly in the browser. We further audit the database indexes and assess DPCexplorer against the **FAIR Principles for Research Software (FAIR4RS)**. The application will soon be accessible at <https://dpcexplorer.areasciencepark.it> and is fully reproducible from the GitHub repository [1] and the associated Zenodo data deposit [2].

Keywords: Protein domain classification, Density Peak Clustering, UniRef50, DPCfam, AlphaFold2, DPCstruct, metacluster, Pfam, Django, PostgreSQL, bioinformatics web application, FAIR data principles, FAIR4RS, Mol*, DPCexplorer.

Contents

Acknowledgments	iv
Abstract	v
1 Introduction	1
1.1 Motivation	1
1.2 Project Overview	2
1.3 Objectives	2
1.4 Thesis Structure	3
2 Background	4
2.1 Proteins and Protein Domains	4
2.1.1 Proteins in a nutshell	4
2.1.2 Protein Domains	5
2.2 Protein Databases Overview	5
2.2.1 Sequence Databases: UniProt	5
2.2.2 Structure Databases	6
2.2.3 Domain Family and Annotation Databases	6
2.3 The Annotation Gap	7
2.4 Density Peak Clustering (DPC)	8
2.4.1 Core Quantities: ρ and δ	8
2.4.2 Centroid Selection: The Decision Graph and γ	9
2.5 DPCfam Pipeline	10
2.5.1 Implementation	10
2.5.2 Validation and Discovery	11
2.6 DPCstruct Pipeline	11
2.6.1 Implementation	11
2.6.2 Validation and Discovery	12
2.7 FAIR Data Principles	13
3 Data Sources, Preprocessing, and Exploratory Analysis	14
3.1 Data Sources	14
3.1.1 DPCfam Dataset	14

3.1.2	DPCstruct Dataset	15
3.2	Project Pipeline	15
3.3	Data Acquisition and Preprocessing	16
3.3.1	Initial Parsing	16
3.3.2	Metadata Consolidation	16
3.4	Exploratory Data Analysis	18
3.4.1	DPCfam Metacluster Properties	19
3.4.2	DPCstruct Metacluster Properties	23
3.4.3	CATH and SCOP Structural Annotations	25
3.5	Key Findings and Their Impact on Design	26
3.6	From CSV to the Web Framework	26
4	System Design and Architecture	27
4.1	Overview	27
4.2	Technology Stack Justification	28
4.3	Django Application Structure	28
4.4	Database Design	28
4.4.1	Schema Overview	29
4.4.2	Performance Optimization	30
4.5	Query Types and Routing	31
4.6	REST API for Data Downloads	33
5	Implementation	34
5.1	Technology Stack	34
5.2	The Landing Page and Search	35
5.3	DPCfam Metacluster Pages	35
5.3.1	The Metacluster List	35
5.3.2	The Metacluster Detail	36
5.4	DPCstruct Metacluster Pages	36
5.4.1	The Metacluster List	36
5.4.2	The Metacluster Detail	37
5.5	Mol* Integration	37
5.5.1	Mol* in a nutshell	37
5.5.2	Why PDBe-Molstar	38
5.5.3	Loading PDBe-Molstar	38
5.5.4	The Django Backend: Preparing PDB File Paths	38
5.5.5	Passing Data to JavaScript	39
5.5.6	Lazy Initialisation and Structure Switching	39
5.6	The Pfam Query	41
5.7	The UniProt Query	42

5.7.1	The Domain Architecture Diagram	42
5.7.2	Toggles and Responsive Layout	42
5.8	The Administration Panel	42
6	Results and Discussion	44
6.1	Application Demonstration	44
6.1.1	Landing Page	44
6.1.2	DPCfam Metacluster Table	45
6.1.3	DPCstruct Metacluster and 3D Viewer	46
6.1.4	Pfam Query Results	48
6.1.5	UniProt Query Results	49
6.2	System Evaluation	50
6.3	FAIR4RS Assessment	51
6.4	Reproducibility	53
7	Conclusion and Future Work	54
7.1	Summary and Contributions	54
7.2	Future Work and Closing Thoughts	55
	Bibliography	56
A	Preprocessing Inventory	60
A.1	Experimental Environment	60
A.2	Inventory	60
B	The PDB File Format	63

Chapter 1

Introduction

1.1 Motivation

Every organism, from a simple bacterium to a human being, depends on proteins to carry out the chemical reactions that make life possible. Understanding what a protein does, in other words, knowing its *function*, is therefore one of the central goals of modern biology [3].

However, there is a growing and serious problem: the number of protein sequences and structures deposited in databases far outpace experts' ability to characterize their functions experimentally. As of 2024, **UniProtKB** [4] contains over 245 million sequences, but only a small fraction of these have been experimentally annotated. The situation with protein structures has become even more dramatic since the publication of **AlphaFold2** [5], which has produced accurate 3D structure predictions for over 200 million proteins, populating the *AlphaFold Protein Structure Database* [6].

One established strategy for narrowing this annotation gap is *protein domain family classification*. Proteins (or their domains) that belong to the same family typically share the same evolutionary origin and, importantly, a similar function. If even one member of a family has a known function, that annotation can be transferred, with appropriate caution, to other members. This is the basic idea behind databases like **Pfam** [7], where proteins are grouped into families based on Multiple Sequence Alignments (MSAs) and profile Hidden Markov Models (HMMs), enabling sensitive detection of distant homologs [8].

The project we describe in this thesis builds on two recent computational biology methods, **DPCfam** [9] and **DPCstruct** [10], that adopt an entirely *unsupervised and automated* approach to protein domain classification. Unlike Pfam and other tools,

which mostly rely on manual expert curation, these methods use the **Density Peak Clustering (DPC)** algorithm [11] to automatically group millions of protein domains into putative families (*metaclusters*) based purely on pairwise similarity scores. The datasets derived from both methods are:

- **DPCfam** [12]: Approximately 23 million sequences from **UniRef50** (v.2017_07) clustered into **81,384** sequence-based metaclusters (46,828 in the Standard subset and 34,556 in DPCfamB).
- **DPCstruct** [13]: Approximately 15 million **AlphaFold** structures (AlphaFold Database v4.0, the Foldseek Clusters representative set) clustered into **28,246** structure-based metaclusters.

Guided by the **FAIR (Findable, Accessible, Interoperable, Reusable) Data Principles** [14], DPCfam [9] and DPCstruct [10] authors established a strong foundation by sharing their results in public **Zenodo** [15] repositories ([12, 13]). However, exploring these raw files still requires significant technical expertise. This thesis contributes to the ongoing **FAIRification** [14] of these datasets by developing a unified **Django-based application**. By bridging the gap between static data files and interactive navigation, we believe our platform could facilitate the downstream analysis and functional annotation of at least one protein through putative domain families identified by the DPC algorithm.

1.2 Project Overview

This thesis describes the design and development of **DPCexplorer**, a unified Django web application for the interactive exploration of the DPCfam and DPCstruct datasets. The application provides a simple, user-friendly interface that allows biologists and data scientists to query the database using familiar biological identifiers (UniProt IDs, Pfam IDs, or DPC metacluster IDs) and to explore the results through metadata tables, downloadable files, and uniquely an integrated 3D protein structure viewer based on **Mol*** [16]. The project was carried out as a six-month internship at the **Laboratory of Data Engineering (LADE)**, Area Science Park, Trieste, Italy, under the supervision of **Dr. Valerio Piomponi** and **Dr. Elaheh Saadat**. It forms the practical component of the Professional Master in Data Management and Curation (MDMC) at SISSA.

1.3 Objectives

The main objectives of the project are:

1. Develop a unified web interface using Django that allows interactive navigation and mining of the DPCfam and DPCstruct datasets.
2. Implement search capabilities for four query types: DPCfam MCID, DPCstruct MCID, Pfam ID (family or clan), and UniProt ID.
3. Enable users to visualize cluster content, including domain metadata, representative sequences and structures, and cross-links to reputable external databases (UniProt, InterPro, AlphaFoldDB, etc.).
4. Integrate the Mol* 3D viewer to enable interactive, in-browser visualization of AlphaFold2-predicted domain structures directly within cluster pages.

1.4 Thesis Structure

The rest of this thesis is organised as follows:

Chapter 2 reviews the biological and computational background: proteins and domains, the main sequence and structure databases, the Density Peak Clustering algorithm, the DPCfam and DPCstruct pipelines, and the FAIR and FAIR4RS principles that frame the project.

Chapter 3 explains how the two Zenodo datasets were acquired, cleaned, and analysed on the ORFEO HPC cluster, and how the exploratory data analysis directly shaped the application’s design.

Chapter 4 presents the technical “blueprint”: the Django Model-View-Template (MVT) architecture, the PostgreSQL schema and its indexes, and the unified routing logic behind the four search types.

Chapter 5 describes how the interface was built: the Django backend, the four-query search, the embedded Mol* viewer for 3D structural exploration, and the read-only-by-default administration panel.

Chapter 6 walks through each query type from the user’s perspective, audits the database indexing strategy, assesses the platform against the FAIR4RS principles, and discusses reproducibility.

Chapter 7 summarises the main contributions and outlines future directions, such as a REST API, an in-browser MSA viewer, and a quantitative FAIR score.

The thesis is supported by two **Appendices (A-B)**, which include a complete inventory of the preprocessing scripts and notebooks, and an overview of the PDB file format, the backbone of protein 3D visualization.

Chapter 2

Background

2.1 Proteins and Protein Domains

2.1.1 Proteins in a nutshell

Proteins are essential biological macromolecules that carry out many of the major functions inside cells [3]. They are built as linear chains of **20 standard amino acids** (see Table B.4 in Appendix), and the order of these amino acids, known as the **primary structure (sequence)**, is encoded by the organism's DNA. Once synthesized, the chain folds into a specific **tertiary structure (three-dimensional shape)**. This folding is driven largely by the chemical properties of the sequence and is often assisted by molecular helper proteins called **chaperones** [3]. A protein's shape strongly influences how it interacts with its environment and, therefore, what biological function it can perform. Because structure and function are tightly linked, even a small change in sequence can sometimes alter folding and impair function [3].

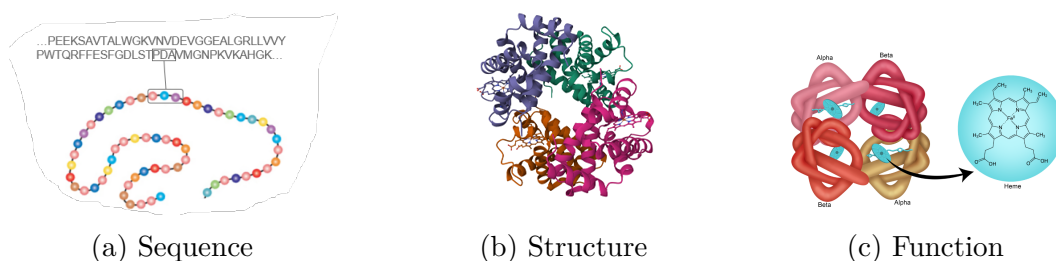


Figure 2.1: A Protein Sequence-Structure-Function Paradigm [17].

Figure 2.1 illustrates a progression from (a) the linear **sequence** of amino acids to (b) the folded 3D **structure**, which ultimately enables the protein's (c) **function**, such as the ability of **hemoglobin** to bind and transport oxygen throughout the body [3].

2.1.2 Protein Domains

Most large proteins are not single, rigid blocks; instead, they are composed of multiple **domains**. These are independent modules that act as the protein’s functional and structural building blocks. A domain is unique because it can fold independently into a stable shape, even if separated from the rest of the protein chain [3].

In evolutionary terms, nature treats these domains like “Lego bricks.” Successful domain designs are reused and combined in different architectures to create new proteins with diverse capabilities [18]. This modularity allows life to evolve complex new functions by reshuffling existing, reliable modules rather than starting from scratch.

2.2 Protein Databases Overview

As technology has enabled the discovery of millions of new sequences and structures, the volume of biological information has exploded. To manage this “protein universe,” researchers rely on specialized databases designed to store, catalog, and annotate sequences, 3D models, and functional data.

2.2.1 Sequence Databases: UniProt

The **Universal Protein Resource (UniProt)**(uniprot.org) [4] is a collaboration between the *European Molecular Biology Laboratory, European Bioinformatics Institute* (EMBL-EBI, UK), the *SIB Swiss Institute of Bioinformatics* (Switzerland), and the *Protein Information Resource* (PIR, USA), serving as the world’s most comprehensive resource for protein sequence and functional data. It is organized into three complementary databases:

1. **UniProtKB (Knowledgebase)**: The central hub for annotated information. It consists of **Swiss-Prot** (manually reviewed, high-quality entries) and **TrEMBL** (automatically annotated, unreviewed entries).
2. **UniRef (Reference Clusters)**: These datasets reduce redundancy by grouping sequences at 100%, 90%, and 50% identity thresholds. UniRef50, for instance, provides an efficient representation of the known protein universe by grouping similar sequences into clusters represented by a single “seed” sequence [18].
3. **UniParc (Archive)**: A non-redundant repository storing the complete history of every unique sequence ever deposited. This ensures data provenance even if a sequence becomes obsolete in its original source [4].

2.2.2 Structure Databases

The **Protein Data Bank (PDB)** (rcsb.org) [19], established in 1971, is the primary global archive for experimentally determined 3D protein structures obtained through X-ray crystallography, NMR spectroscopy, or cryo-electron microscopy (cryo-EM). Each entry is stored as a `.pdb` or `.mmCIF` file containing the 3D coordinates of every atom.

While the PDB currently holds over 220,000 structures, it covers only a tiny fraction of the known protein sequence space. To bridge this gap, the **AlphaFold Protein Structure Database (AlphaFoldDB)** (alphafold.ebi.ac.uk) [5, 6] was launched in 2021. It provides high-accuracy predicted structures for over 214 million proteins, encompassing nearly all entries in UniProtKB. To manage this vast volume of data, the **Foldseek Cluster** database (cluster.foldseek.com) [20] was created by clustering the entire AlphaFoldDB at 50% sequence similarity and 90% structural overlap, resulting in a non-redundant set of approximately 15 million representative proteins.

2.2.3 Domain Family and Annotation Databases

The **Protein family database (Pfam)** [7] is a widely used resource that organizes protein regions into families based on conserved sequence patterns using profile Hidden Markov Models (HMMs). As of release 37.0 (2024), it contains **21,979** families and **709** clans. A major recent advancement is **Pfam-N** [8], which uses deep learning architectures, specifically *transformer-based models* (Maskformer) and *convolutional neural networks* to detect distant homologs, increasing UniProtKB coverage by 8.8%.

InterPro (ebi.ac.uk/interpro) [21] serves as a unified hub that integrates protein “signatures” from Pfam and other specialized databases to power the automatic annotation of UniProtKB. Following the retirement of the original Pfam website in 2022, InterPro has become the official portal for exploring Pfam’s extensive family data [8].

While sequence-based tools are vital, structural classification databases organize the “protein universe” based on three-dimensional folds:

- **CATH** (cathdb.info) [22] partitions structures into domains and classifies them into a four-level hierarchy: **C**lass, **A**rchitecture, **T**opology, and **H**omologous superfamily.
- The **Structural Classification of Proteins (SCOP)** (scop.mrc-lmb.cam.ac.uk) [23] relies on expert manual curation to identify distant evolutionary relationships that automated methods might overlook.
- The **Evolutionary Classification of protein Domains (ECOD)** [24] (prodata.swmed.edu/ecod/) prioritizes an **evolutionary perspective**, grouping

domains that share a common ancestor even when their structural similarity has significantly diverged.

2.3 The Annotation Gap

Advances in sequencing technology have made it easy to map the primary sequences of millions of proteins, but **experimental characterisation remains a painstakingly slow process** [9]. This discrepancy has created a massive “**annotation gap**,” which represents one of the central challenges in modern biology [10]. Since protein domains are evolutionarily conserved, methods such as *homology-based inference* can predict a protein’s function based on its known relatives [9]. However, as the volume of data explodes (Figure 2.2), manual curation can no longer keep pace with the discovery of new, divergent sequences [18].

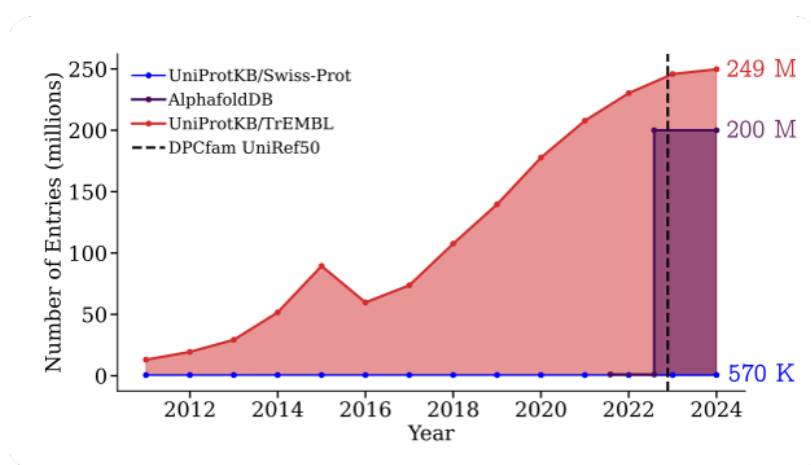


Figure 2.2: Yearly growth of the UniProt Knowledgebase [17, 25].

Figure 2.2 illustrates the dramatic scaling of unreviewed sequences (**TrEMBL**) compared to the stable, linear growth of reviewed entries (**Swiss-Prot**). Over the last decade, TrEMBL entries have surged to nearly **250 million**, while the high-quality, expert-curated Swiss-Prot database has remained relatively flat at approximately **570,000**. The temporary decline in TrEMBL records visible in 2015 resulted from new sequence de-duplication protocols designed to reduce redundancy in bacterial proteomes [25].

To bridge this gap, computational tools capable of automatically organizing the “protein universe” are essential. At the heart of this solution is the **Density Peak Clustering (DPC)** algorithm [11]. Recently applied to both protein sequences (**UniRef50**) and three-dimensional structures (**AlphaFoldDB**), DPC has classified domains into broad putative families known as “**Metaclusters**.” These metaclusters act as functional hypotheses, “*turning the lights on*” for thousands of novel protein families that manual curation has yet to reach [9, 10].

2.4 Density Peak Clustering (DPC)

Clustering is a fundamental task in *unsupervised machine learning* that involves partitioning a dataset into meaningful groups based on similarity without the need for prior labels. However, traditional methods often face significant limitations. For example, **K-means** [26] requires the *number of clusters to be defined in advance and assumes they are spherical*, which rarely reflects the complexity of real-world data. While density-based approaches like **DBSCAN** [27] can handle *arbitrary shapes*, they frequently struggle with varying densities and ambiguous boundaries.

The Density Peak Clustering (DPC) algorithm, introduced by **Rodriguez and Laio** in 2014 [11], provides a more flexible alternative. It is built on two intuitive principles:

1. **Cluster centers have a high local density**, meaning they are surrounded by a large number of neighbors.
2. **Cluster centers are located at a relatively large distance** from any other points with a higher density.

The main steps of the algorithm are described below:

2.4.1 Core Quantities: ρ and δ

For a dataset of N points with pairwise distances $\{d_{ij}\}$, the DPC algorithm computes two essential values for each point i :

1. **Local Density** (ρ_i): This value represents the number of neighbors within a cut-off distance d_c :

$$\rho_i = \sum_{j \neq i} \chi(d_{ij} - d_c), \text{ where } \chi(x) = \begin{cases} 1, & x < 0 \\ 0, & \text{otherwise.} \end{cases} \quad (2.1)$$

Alternatively, a Gaussian kernel $\rho_i = \sum_{j \neq i} \exp(-(d_{ij}/d_c)^2)$ can be used for smoother estimates in smaller datasets.

Rule of thumb: Set d_c such that the average number of neighbors per point is approximately **1–2%** of the total dataset [11].

2. **Minimum Distance to a Denser Point** (δ_i): This measures the distance to the closest neighbor that has a strictly higher local density:

$$\delta_i = \min_{j: \rho_j > \rho_i} (d_{ij}) \quad (2.2)$$

For the point with the highest overall density, δ_i is conventionally set to the maximum distance found in the dataset ($\delta_{\max} = \max_j d_{ij}$) to ensure it remains an outlier.

2.4.2 Centroid Selection: The Decision Graph and γ

The **Decision Graph** plots δ_i (y-axis) against ρ_i (x-axis). Because cluster centers are local density maxima far from other dense regions, they appear as prominent outliers in the upper-right corner of the graph. To automate the selection of these peaks, the γ score is defined as the product of density and isolation:

$$\gamma_i = \rho_i \cdot \delta_i. \quad (2.3)$$

True cluster centers are identified by sorting all points by γ in decreasing order and searching for a **significant numerical gap** in the ranking. Points above this gap are selected as the k centroids. Once centroids are chosen, every other point is assigned to the same cluster as its nearest neighbor of higher density in a single, non-iterative step.

Algorithm 1 Density Peak Clustering (DPC)

Require: Dataset $\mathcal{X} = \{x_1, \dots, x_N\}$, distance metric $d(\cdot, \cdot)$, cut-off d_c

Ensure: Cluster labels $\{\ell_1, \dots, \ell_N\}$

- 1: **Precompute:** Calculate all pairwise distances d_{ij}
 - 2: **Local Density:** Compute ρ_i for all i (Eq. 2.1)
 - 3: **Isolation:** Compute δ_i for all i (Eq. 2.2)
 - 4: *Convention: For the point with highest density, set $\delta_i = \max_j d_{ij}$*
 - 5: **Peak Selection:**
 - 6: Calculate decision value γ_i (Eq. 2.3) for all points i
 - 7: Identify centroids $\mathcal{C}$ by looking for outliers in the γ distribution
 - 8: **Initial Labels:** Assign a unique label ℓ_c for each centroid $c \in \mathcal{C}$
 - 9: **Clustering:** Sort all non-peak points by decreasing density ρ
 - 10: **for** each point $i \notin \mathcal{C}$ in decreasing ρ order **do**
 - 11: $\ell_i \leftarrow \ell_{nn(i)}$, where $nn(i)$ is the nearest neighbor with higher density
 - 12: **end for**
 - 13: **return** $\{\ell_i\}$
-

Algorithm 1 identifies clusters without requiring prior assumptions about their *number or shape* [25]. Peak selection and cutoff distance (d_c) are the two key *hyperparameters* for tuning this algorithm. This unsupervised approach serves as the foundation for both DPCfam [9], which operates in *protein sequence space*, and DPCstruct [10], which operates in *protein structure space* to automatically organize millions of protein domains

into putative families called *metacusters*. We summarize their key steps in this thesis by presenting their pipelines. For an optimal exploration of the mathematics and the High Performance Computing (HPC) strategies (e.g., matrix parallelization) behind them, we refer the reader to the papers [9] and [10] or the thesis [25].

2.5 DPCfam Pipeline

DPCfam [9] is a massive, unsupervised classification of protein domains based on sequence similarity. It was designed to bridge the annotation gap by automatically grouping sequences into families without relying on manual expert curation. The pipeline uses **UniRef50 (v.2017_07)** as its input, which contains approximately **23 million sequences** that share no more than 50% identity (see subsection 2.2.1).

2.5.1 Implementation

The pipeline transforms raw sequence data into organised families through four major stages, as illustrated in Figure 2.3:

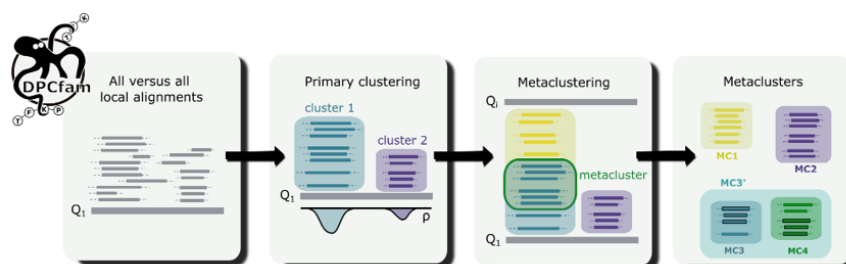


Figure 2.3: Overview of the DPCfam sequence clustering pipeline. The process uses BLASTp for all-versus-all local alignments of 23 million sequences, followed by a two-step Density Peak Clustering approach: primary clustering segments individual proteins into domain-like units, while metaclustering groups these units into putative families [25].

1. **All-versus-all Local Alignment:** Using **BLASTp**, the algorithm compares the 23 million sequences against one another to identify similar regions. To manage this enormous task, the authors performed approximately **100,000** batch searches, with each batch containing **200** query sequences.
2. **Primary Clustering:** For every individual protein, the resulting alignments are grouped via the DPC algorithm based on where they overlap on that protein's sequence. This segments the protein into domain-like units called *primary clusters*.
3. **Metaclustering:** The DPC algorithm is then used to compare these primary clusters across the entire database. Clusters that share a significant number of common alignments are grouped into *metacusters*.

4. **Merging and Filtering:** To reduce redundancy, similar metaclusters are merged, and outliers are removed to ensure high-quality “seed” families.

2.5.2 Validation and Discovery

To ensure the results were biologically meaningful, the authors compared DPCfam against the expert-curated **Pfam** and **ECOD** databases. The algorithm successfully recovered approximately **81% of medium-to-large Pfam families** and **72% of ECOD families**, demonstrating that unsupervised clustering can accurately capture established evolutionary relationships.

Beyond matching known data, the pipeline identified metaclusters that completely lack a Pfam mapping: **14,190** in the Standard subset and **18,989** in DPCfamB. These represent a significant pool of *putative novel protein families* [9]. To validate these findings, the authors collaborated with the Pfam team, who adopted **63** of these unknown clusters as official new families in **Pfam release 35.0** (for example, MC202620 → PF20147 and MC15137 → PF20146). This discovery highlights the potential of DPCfam to uncover the “dark matter” of the protein universe [25].

2.6 DPCstruct Pipeline

DPCstruct [10] serves as the structural counterpart to DPCfam. Because protein structures are often more evolutionarily conserved than sequences, this pipeline is essential for uncovering **remote homology**: identifying relationships between proteins that share less than 20% sequence identity.

2.6.1 Implementation

The **DPCstruct** pipeline, illustrated in Figure 2.4, is a high-performance adaptation of the DPCfam logic, specifically re-engineered to handle 3D structural data through four primary stages:

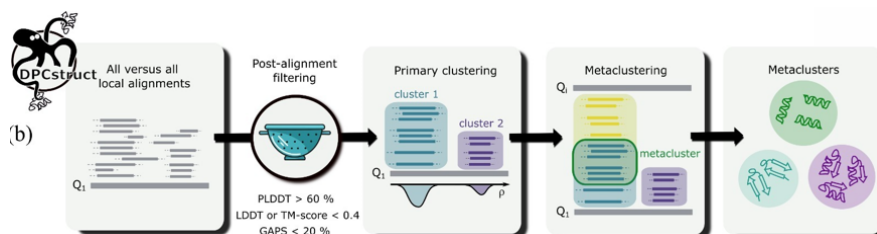


Figure 2.4: Overview of the DPCstruct structural clustering pipeline. After aligning 15 million AlphaFold structures with Foldseek, the pipeline applies quality filters (pLDDT, TM-score, and gap thresholds) before using DPC to identify structural domains and group them into non-redundant metaclusters representing conserved protein folds [25].

1. **Input Data:** The process starts with **15 million representative structures** from the Foldseek Clusters database, which provides a non-redundant view of the structural diversity in AlphaFoldDB (see subsection 2.2.2).
2. **Structural Alignment (Foldseek & MMseqs2):** Instead of sequence-based tools like BLAST, the pipeline performs all-versus-all local alignments using **Foldseek** [20]. Foldseek “translates” 3D shapes into a structural alphabet, which allows it to leverage the highly optimized **MMseqs2** engine. This approach transforms structural alignment into a sequence-based problem, enabling searches that are four to five orders of magnitude faster than traditional methods while maintaining high sensitivity [25].
3. **Post-Alignment Filtering:** To handle the complexity of 3D data and ensure biological reliability, a rigorous filtering stage is applied. Alignments are discarded unless they meet three criteria: an average **pLDDT** ≥ 60 (ensuring high-confidence structural predictions), a **TM-score or LDDT** ≥ 0.4 (confirming global or local shape similarity), and a Gap threshold of $< 20\%$.
4. **Two-Step DPC Engine:** The core clustering follows a refined two-stage Density Peak Clustering process:
 - (a) **Primary Clustering:** Local alignments for each protein are grouped into putative domains. To ensure these clusters are both recurrent and distinct, the pipeline employs a double-threshold criterion: a **density** (ρ) ≥ 10 (requiring at least 10 overlapping alignments) and an **isolation** (δ) ≥ 0.4 (ensuring the cluster center is well-separated from other high-density regions).
 - (b) **Metaclustering:** These domains are then grouped globally based on their shared structural elements into **28,246 structural metaclusters**.

2.6.2 Validation and Discovery

To ensure DPCstruct produces reliable biological data, the authors compared it against established, expert-curated standards. The algorithm successfully recovered **91% of SCOP folds and 83% of CATH folds**, confirming that it can automatically identify known protein shapes with high accuracy. The comparison with the sequence-based **Pfam** database, at the clan level, was even more revealing. For this analysis, they focused on **14,423 metaclusters** (approximately 51% of the total) that contained at least two structures with a known Pfam annotation. Among these, **70% achieved a perfect consistency score of 1**, meaning every member of those clusters shared the exact same Pfam label. Beyond matching known data, the algorithm also serves as a powerful discovery tool. Notably, **24% of the metaclusters** show no significant

similarity to any existing database, including Pfam, CATH, or SCOP. These “unknown” clusters exhibit the same high structural quality as the known ones, pointing toward the discovery of *potentially novel protein folds*.

To conclude this chapter, we address the framework used to ensure these results are shared effectively. The scientific value of these metaclusters depends entirely on their **usability** and **accessibility** to the wider research community.

2.7 FAIR Data Principles

The **FAIR Data Principles**, formally published in 2016 [14], state that scientific data should be **F**indable, **A**ccessible, **I**nteroperable, and **R**eusable, by both humans and machines. The upstream **DPCfam** and **DPCstruct** datasets already meet them at the data level: both are published on **Zenodo** [15, 12, 13] with persistent DOIs, open downloads, standard identifiers and formats, and documented generation pipelines.

Our own contribution, however, is **research software**, not a dataset. We therefore evaluate it against the **FAIR Principles for Research Software (FAIR4RS)** [28], which keep the four familiar letters but add software-specific sub-criteria: a citable version DOI, an open licence, declared dependencies, and qualified links to the digital objects the software depends on. The full FAIR4RS assessment of DPCexplorer is presented in Chapter 6.

What was still missing was a way for non-technical users to explore this data without downloading and parsing **gigabyte-scale archives**. That is the gap our project fills: a web application that turns static Zenodo files into an interactive, searchable resource.

In the next chapter, we explain how we acquired and preprocessed the two datasets before their ingestion into our application’s database.

Chapter 3

Data Sources, Preprocessing, and Exploratory Analysis

This chapter tells the story of how raw data files sitting on **Zenodo** [15] became a clean, queryable database ready to power a web application. The journey has five stages, illustrated in Figure 3.1, but this chapter focuses on the first two: where we got the data and how we prepared it. The remaining three stages: Database design, backend logic, and the user interface are covered in Chapters 4 and 5.

3.1 Data Sources

The project relies on two publicly available datasets, both hosted on Zenodo [12, 13].

3.1.1 DPCfam Dataset

The DPCfam repository [12] is divided into two subsets based on how many seed sequences each metacluster contains: the **Standard** set (≥ 50 seeds per metacluster) and **DPCfamB** (25 to 49 seeds per metacluster). The key files are:

- **Metadata and Seeds** ([metaclusters_xml.tar.gz](#)) : XML tables containing unaligned seed sequences and statistical metrics such as cluster size, average length, and low-complexity fractions for each metacluster.
- **Model Profiles** ([msas](#) and [hmms.tar.gz](#)) : Multiple Sequence Alignments (MSAs) and Profile Hidden Markov Models (HMMs) used for family classification.
- **Annotated Database** ([UniRef50_annotated_xml.tar.gz](#)) : An 81 GB XML file mapping the full **UniRef50** database to both **Pfam** families and DPCfam metaclusters.

3.1.2 DPCstruct Dataset

The DPCstruct repository [13] covers approximately 15 million protein structures from the **AlphaFold Database (v4.0)**. Its key files are:

- **Structural Classification** ([dpcstruct_classification.tsv](#)) : A master list of 1,642,061 identified structural domains, mapping UniProt IDs and residue coordinate ranges to specific metaclusters.
- **Metacluster Analytics** ([mcs_properties.tsv](#)) : Statistical profiles for 28,246 structural metaclusters, including AlphaFold2 confidence (pLDDT) and structural consistency metrics (TM-scores, IDDT), etc.
- **Representative Models** ([mcs_reps.fasta](#) and [mcs_reps_pdbs.zip](#)) : FASTA sequences and 3D PDB coordinates for the cluster centre or the highest-pLDDT representative of each metacluster.
- **Validation Files** Structural alignment results against **Pfam 36.0**, **CATH**, and **SCOP** classifications, used to assess the biological coherence of the metaclusters.

3.2 Project Pipeline

In order to transform these static Zenodo archives into an interactive web experience, we developed a five-stage end-to-end pipeline illustrated in Figure 3.1.

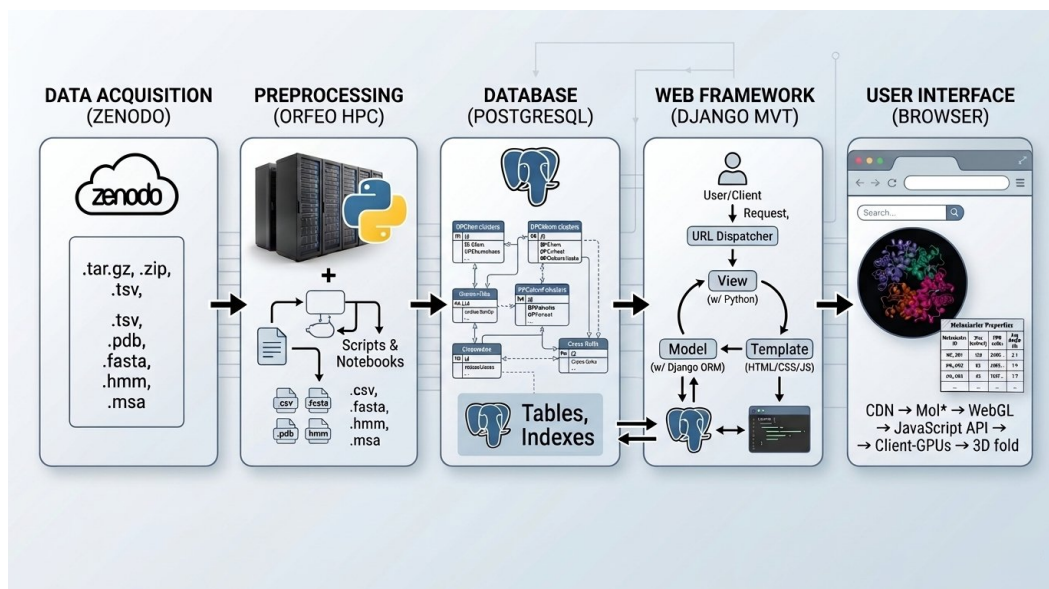


Figure 3.1: Full project pipeline. Raw datasets are retrieved from Zenodo and preprocessed using Python notebooks and scripts on the ORFEO HPC cluster. The refined data is then split: relational metadata populates a PostgreSQL database, while sequence and structural files (FASTA, PDB, MSA, HMM) are stored as static assets. A Django backend serves this data to a web interface featuring an integrated Mol* 3D viewer.

1. **Data Acquisition:** download of all raw files from Zenodo [15, 12, 13].
2. **Preprocessing:** parsing, cleaning, and consolidating raw files into CSV tables using Python notebooks and scripts run on the ORFEO HPC cluster.
3. **Database Ingestion:** loading the processed CSV files into PostgreSQL, with indexes tuned for fast lookup by Protein ID, Pfam ID, or Metacluster ID.
4. **Backend Logic:** Django views that translate user queries into database calls and package the results for the interface.
5. **User Presentation:** interactive metadata tables and in-browser 3D visualisation via the Mol* viewer.

3.3 Data Acquisition and Preprocessing

Due to the size of some source files exceeding the memory capacity of a standard workstation (the annotated UniRef50 XML alone is 81 GB), the whole preprocessing was carried out on the **ORFEO High-Performance Computing (HPC)** cluster (see hardware specifications in Table A.1, Appendix A). A complete inventory of every notebook and script is provided in Table A.2 while their corresponding files are located in the `static/` directory of the project repository [1].

3.3.1 Initial Parsing

Both datasets were downloaded directly from Zenodo [15]. For DPCfam [12], the authors provided a BASH script (`parse.sh`) that converts the raw XML archives into space-separated value (SSV) files and per-metacluster FASTA files with seed sequences. These SSV files are the starting point for all subsequent DPCfam work. For DPCstruct [13], the metadata was already in TSV format and needed no initial conversion.

3.3.2 Metadata Consolidation

DPCfam - metacluster properties. Each SSV file was first explored individually in a dedicated notebook (see Table A.2). We then merged all relevant fields into a single consistent DataFrame using `dpcfam_all_mcs_props.ipynb`, processing Standard DPCfam first, then DPCfamB, and finally combining the two. The result is the **81,384-row** master properties table described in Section 3.4.

DPCfam - seed sequences. `dpcfam_all_mcs_sequences.ipynb` merged all parsed FASTA files into a single `dpcfam_all_mcs_sequences.csv`, where each row represents

one domain from a specific protein within a specific metacluster. This file powers the **Sequences** tab on every DPCfam metacluster page.

DPCfam - Pfam and protein data from the 81 GB XML. Loading the full annotated XML into memory is not feasible. We therefore wrote [extract_uniref_data.py](#), which processes the file with a single streaming pass and extracts three outputs, subsequently organized using [exploring_uniref50_data.ipynb](#):

- `uniref50_proteins.csv`: every UniProt accession present in UniRef50 together with its protein length in amino acids.
- `uniref50_pfam_valid.csv`: Pfam family IDs and their coordinate ranges on each protein.
- `uniref50_dpcfam_valid.csv`: DPCfam domain coordinate ranges per protein.

DPCstruct - metacluster properties. The `mcs_properties.tsv` and `dpcstruct_consistency.tsv` files were merged in [dpcstruct_mcs_properties.ipynb](#), producing the 28,246-row `dpcstruct_mcs_properties.csv` described in Section 3.4.

DPCstruct - structural domain sequences. [dpcstruct_mcs_sequences_info.ipynb](#) read `dpcstruct_classification.tsv` to collect metacluster IDs, protein accessions, and domain ranges, then joined in the representative sequences from `mcs_reps.fasta`. The notebook also organized sequences into individual per-metacluster `.fasta` files, which are available for download on each metacluster detail page.

DPCstruct - organising PDB files. The original `mcs_reps_pdb/` directory was a flat folder containing thousands of PDB files. We used the [dpcstruct_reps_pdb.ipynb](#) notebook to reorganize them into `MCID_pdb/` subfolders. This precise structure is what the Django backend needs to serve the PDB files to the Mol* viewer (see section 5.5). Finally, the notebook packages these subfolders into individual `.zip` files, allowing users to download them directly from each metacluster's detail page.

DPCstruct - resolving protein lengths. Unlike DPCfam, the DPCstruct Zenodo release provides no protein lengths. These are required for the domain-architecture diagram on the protein detail page. Across 1,642,061 structural domains, we identified **1,252,179 unique proteins** in need of length data. We resolved them through a four-step cascade:

1. **Cross-reference with DPCfam.** Every protein already present in our UniRef50 table got its length from there. This resolved **316,461 proteins** ($\approx 25.27\%$).

2. **UniProt REST API (UniProtKB).** [fetch_uniprot_complete.py](#) queried the live UniProt API for the remaining **935,718 proteins**. The API worked for available proteins but silently skipped removed or obsolete entries.
3. **Swiss-Prot and TrEMBL FASTA releases.** We downloaded the full UniProtKB release (last modified on **2026-01-28 23:00**), containing **Swiss-Prot** (~89 MB compressed) and **TrEMBL** (~49 GB compressed), and parsed them with [extract_lengths_from_fasta.py](#) to capture any proteins the API had missed.
4. **UniProt REST API (UniParc).** The entries still missing (around 30% of the total) were proteins that had been removed from UniProtKB but are preserved in the **UniParc** archive. [fetch_missing_lengths.py](#) queried UniParc and successfully retrieved all remaining lengths.

The four outputs were merged by [dpcstruct_all_proteins_lengths.ipynb](#) into `dpcstruct_all_proteins_lengths.csv`. A final notebook, [dpc_protein_lengths.ipynb](#), then combined DPCfam and DPCstruct protein data into the master reference file `dpc_protein_lengths.csv`. This file has two jobs: validating UniProt ID queries at search time and supplying lengths to the protein detail page.

Master Pfam reference. [dpc_pfam_ids.ipynb](#) collected every unique Pfam family and clan ID appearing across both datasets and produced `dpc_pfam_ids.csv`, with columns `pfam_id` and `pfam_type` (Family or Clan). This file is the gatekeeper for Pfam ID searches: if an ID is not in this table, the application rejects it immediately.

CATH and SCOP annotation files. [dpcstruct_explore_cath_in_depth.ipynb](#) read both CATH annotation files provided with DPCstruct, restricted the analysis to the stricter `qc=0.8` dataset, and produced `cleaned_annotated_cath_qc0.8_t0.5_10.5.csv`. The same logic was applied to SCOP via [dpcstruct_explore_scop_in_depth.ipynb](#), producing `cleaned_annotated_scop_qc0.8_t0.5_10.5.csv`. Both files display structural consistency data on the DPCstruct Metaclusters home page.

3.4 Exploratory Data Analysis

Once all CSV files were ready, we ran a systematic Exploratory Data Analysis (EDA) using **ydata-profiling (v4.19.1)** [29], which generates interactive HTML reports covering column distributions, missing value patterns, and pairwise correlations. The ten core files produced by Section 3.3 are listed in Table 3.1 and are accessible at [2].

Table 3.1: Core preprocessed CSV files ingested into the application [2].

ID	Filename	Content
1	dpcfam_all_mcs_props.csv	DPCfam metacluster properties and Pfam annotations (81,384 rows)
2	dpcfam_all_mcs_sequences.csv	DPCfam domain sequences per metacluster (16,602,778 rows)
3	alphafold_dpcfam_reps.csv	AlphaFold representatives for DPCfam metaclusters (38,668 rows)
4	dpcstruct_mcs_properties.csv	DPCstruct metacluster properties and Pfam annotations (28,246 rows)
5	dpcstruct_mcs_sequences.csv	DPCstruct domain sequences and representatives (1,642,061 rows)
6	cleaned_annotated_cath_qc0.8_t0.5_l0.5.csv	DPCstruct CATH fold annotations ($QC = 0.8$, $TM = 0.5$, $L = 0.5$, 1,274 rows)
7	cleaned_annotated_scop_qc0.8_t0.5_l0.5.csv	DPCstruct SCOP fold annotations ($QC = 0.8$, $TM = 0.5$, $L = 0.5$, 1,359 rows)
8	uniref50_pfam_valid.csv	Pfam IDs and domain coordinate ranges per protein (14,554,301 rows)
9	dpc_pfam_ids.csv	Unique Pfam IDs across both datasets (19,352 rows)
10	dpc_protein_lengths.csv	Master protein length registry (14,112,858 rows)

We profiled most of the files before any database loading. This section focuses on the two metacluster properties tables and the structural annotation files, as these are the most complex and had the greatest impact on application design. Full interactive reports are available in the project repository under `static/profiling_reports/`, and can be downloaded for interactive exploration in any web browser.

3.4.1 DPCfam Metacluster Properties

DataFrame in a nutshell

The final DPCfam properties file merges **Standard DPCfam** (46,828 metaclusters, ≥ 50 seeds) with **DPCFamB** (34,556 metaclusters, 25 - 49 seeds), giving **81,384 rows and 12 columns**. After cleaning (described below), the file contains zero missing values and zero duplicate rows. Table 3.2 describes each column.

Table 3.2: Column descriptions for the DPCfam metacluster properties file.

Column	Description
mcid	Unique metacluster identifier (e.g. MC1)
size_uniref50	Number of UniRef50 seed sequences in the metacluster
avg_len	Mean domain length of the seeds (in amino acids)
std_avg_len	Standard deviation of seed domain lengths
lc_percent	% of residues located in low-complexity regions
cc_percent	% of residues located in coiled-coil regions
dis_percent	% of residues predicted as intrinsically disordered
tm	Average number of transmembrane regions per domain
pfam_da	Dominant Pfam Architecture: the most frequent Pfam annotation among seeds; tagged UNKNOWN when no seed carries a Pfam label
da_percent	% of seeds matching the Dominant Architecture
avg_ov_percent	Average % boundary overlap between the metacluster and Pfam, computed only for seeds that match the DA
overlap_label	Qualitative overlap category (see Figure 3.2)

Data Cleaning

Standard DPCfam. An initial profiling run on the 46,828-row Standard file revealed **42,570 missing cells** (7.6%), all concentrated in exactly three columns: `da_percent`, `avg_ov_percent`, and `overlap_label`, each with **14,190 missing values** (30.3%). The identical count across all three was the first clue. Investigating further confirmed our hypothesis: every missing row was one where `pfam_da` = "UNK", meaning there is no Pfam annotation to measure anything against. The fix was straightforward and logically consistent: rename "UNK" to "UNKNOWN", then fill the three dependent columns with `da_percent` = 0, `avg_ov_percent` = 0, and `overlap_label` = "NONE". Zero duplicate rows were found.

DPCfamB. The 34,556-row DPCfamB file initially showed no missing values, which sounded like good news. However, a deeper check revealed an internal inconsistency: **18,989 rows** (55% of DPCfamB) where `pfam_da` = "UNK" had `da_percent` = 100, `avg_ov_percent` = 0, and `overlap_label` = "shifted"; values that directly contradict the logic of the Standard set. We applied the same correction: when `pfam_da` = "UNKNOWN", `da_percent` = `avg_ov_percent` = 0, and `overlap_label` = "NONE".

Merged file. After merging Standard and DPCfamB, we also standardised the `pfam_da` column: concatenated Pfam IDs such as PF00457PF00123 were reformatted

with a “-” delimiter to produce PF00457-PF00123, a format that the application can parse reliably.

Univariate Analysis

Metacluster size. Most metaclusters are small: the median is just 58 sequences, but a few very large, ancient domain families raise the mean to 204 and create a strong right skew in the distribution ($\gamma_1 = 43.6$), with the largest metacluster reaching 131,827 members.

Domain length. Lengths range from 50 to 3,211 amino acids, with a mean of 177.8 aa. The distribution is also right-skewed, though much less dramatically: proteins tend to keep individual domains short and compact, so there are far fewer extreme outliers than in metacluster size.

Sequence composition. Coiled-coil regions are rare; 77.8% of metaclusters have none. Disordered regions are common: on average, 20.7% of residues are disordered, and only 3.3% of metaclusters are fully ordered. Most domains are not membrane-embedded: 62.7% do not show any transmembrane signal.

The `overlap_label` and Dark Matter

The `overlap_label` column is the most biologically revealing field. It describes how the boundary of a DPC-defined domain aligns with its closest Pfam family, and takes one of five values illustrated in Figure 3.2.



Figure 3.2: Pfam overlap label categories [25]. *Equivalent*: DPC and Pfam boundaries agree closely. *Reduced*, *Extended*, *Shifted*: three types of boundary mismatch. *NONE*: no Pfam annotation exists for this metacluster: these are **dark matter**.

Figure 3.3 shows the distribution of these labels across the three datasets.

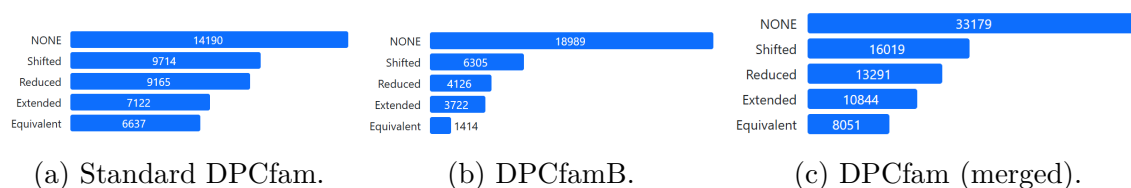


Figure 3.3: Distribution of `overlap_label` across DPCfam subsets. Dark matter (**NONE**) is the single largest category, particularly in DPCfamB.

The contrast between the two subsets is striking. In Standard DPCfam, 30.3% of metaclusters are dark matter and 14.2% are classified as *Equivalent*. In DPCfamB, dark matter jumps to 55.0% while *Equivalent* collapses to just 4.1%. This observation directly informed our decision to implement a three-way filter (**DPCfam / Standard DPCfam / DPCfamB**) in the web application’s metacluster table view, so that users can restrict results to the more reliable Standard subset when annotation quality matters.

Correlation Analysis

Figure 3.4 shows the Pearson correlation matrix for all numeric columns.

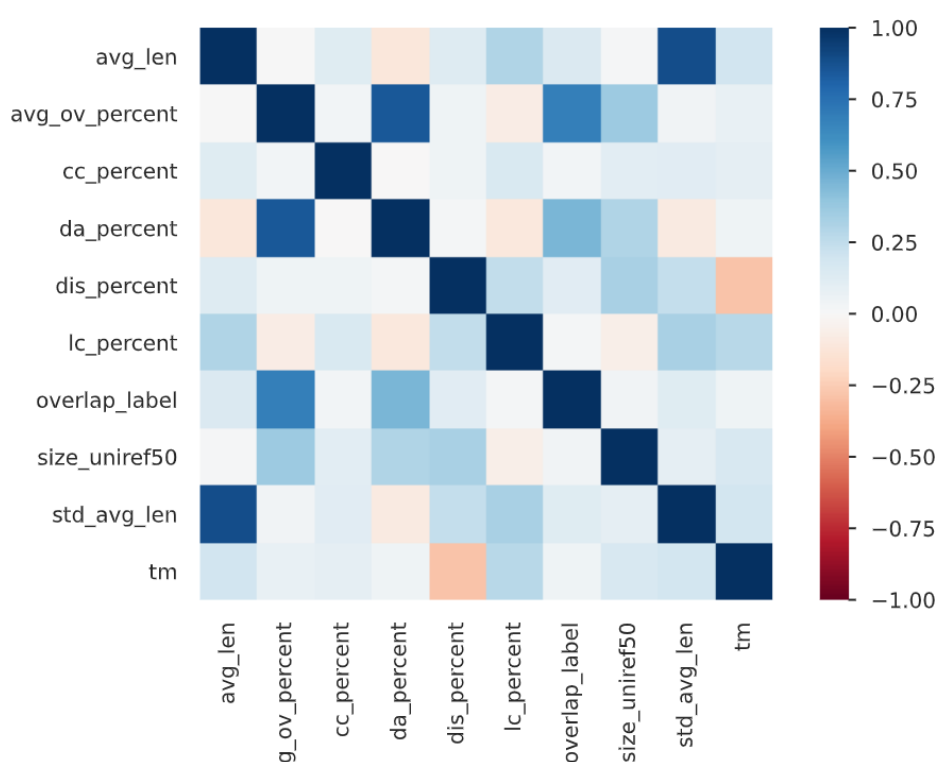


Figure 3.4: DPCfam Pearson correlation matrix. Dark blue = strong positive correlation; dark red = strong negative correlation.

Three patterns stand out. First, `avg_len` and `std_avg_len` are tightly coupled: longer domains naturally display more internal length variability, simply because there is more sequence room for divergence. Second, `da_percent` and `avg_ov_percent` are nearly redundant: when no seeds match a Pfam family, both are zero; when many do, both scale together. Third, and most importantly, `size_uniref50` is essentially uncorrelated with any annotation metric. A large metacluster is not necessarily well-annotated, and a small one is not necessarily dark matter. Annotation quality in DPCfam is determined by evolutionary history, not family size.

3.4.2 DPCstruct Metacluster Properties

DataFrame in a nutshell

After merging the properties and Pfam consistency files, we obtained **28,246 rows and 12 columns**, with zero missing values and zero duplicate rows after cleaning. Table 3.3 describes each column.

Table 3.3: Column descriptions for the DPCstruct metacluster properties file.

Column	Description
<code>mc_id</code>	Unique metacluster identifier (e.g. MCO)
<code>mc_size</code>	Number of structural domain members
<code>len_aa</code>	Mean domain length of all members (amino acids)
<code>len_std</code>	Standard deviation of domain lengths
<code>len_ratio</code>	Length coefficient of variation
<code>plddt</code>	Mean AlphaFold2 confidence score (pLDDT) across all members
<code>disorder</code>	Mean fraction of residues predicted as intrinsically disordered
<code>tmscore</code>	Mean intra-cluster length-normalized TM-Score (structural consistency)
<code>lddt</code>	Mean intra-cluster normalized IDDT score (local structural similarity)
<code>pidnt</code>	Mean intra-cluster sequence identity (%)
<code>pfam_score</code>	Global Pfam score quantifying the agreement between the metacluster and Pfam.
<code>pfam_da</code>	Assigned Pfam labels (clans or families); UNKNOWN when absent

Data Cleaning

The raw merge produced **27,646 missing cells** (7.0%), all from the Pfam columns: **13,823 metaclusters** (48.9%) had no Pfam annotation whatsoever (see subsection 2.6.2). We applied the same imputation logic used for DPCfam: `pfam_score` = 0 and `pfam_da` = "UNKNOWN" for all unannotated rows.

For the sequences file (1,642,061 rows), **1,585,623 cells** in the `prot_seq` column are intentionally NaN: only two representative sequences per metacluster are stored, and the rest of the classification rows are left without sequence data by design. We kept these as NaN in the database and display an informative message on the application side. Zero duplicate rows were found in either file.

For the CATH annotations file (1,274 rows), **116 cells** in the `lddt` column were missing (9.1% of that column); we filled them with zero. The SCOP file (1,359 rows) had no missing values.

Univariate Analysis

Metacluster size. The size distribution is again severely right-skewed ($\gamma_1 = 38.9$): mean = 58.1 members, median = 16, maximum = 30,643. This mirrors the DPCfam pattern: a few large families dominate while the majority of families are small.

AlphaFold confidence. The mean pLDDT across all members is **82.5** (IQR: 77.97 - 87.64), comfortably above the threshold of 70 that AlphaFold2 considers “confident”.

Structural consistency. The mean intra-cluster TM-score is **0.699** (range: 0.16 - 1.0), well above the 0.5 threshold used in the literature to define a shared structural fold.

Sequence identity. The mean intra-cluster sequence identity is only **34.5%**, placing most metaclusters deep inside the **twilight zone** of sequence comparison, where standard alignment methods become unreliable. This is DPCstruct’s core contribution: it groups proteins that share a fold even when their sequences have diverged to the point of being unrecognisable by sequence-based tools.

Dark matter. `pfam_score` = 0 in **49.3%** of all metaclusters (13,931 out of 28,246). Roughly half of DPCstruct’s families have no counterpart in Pfam, making them probable novel structural families.

Correlation Analysis

Figure 3.5 shows the Pearson correlation matrix.

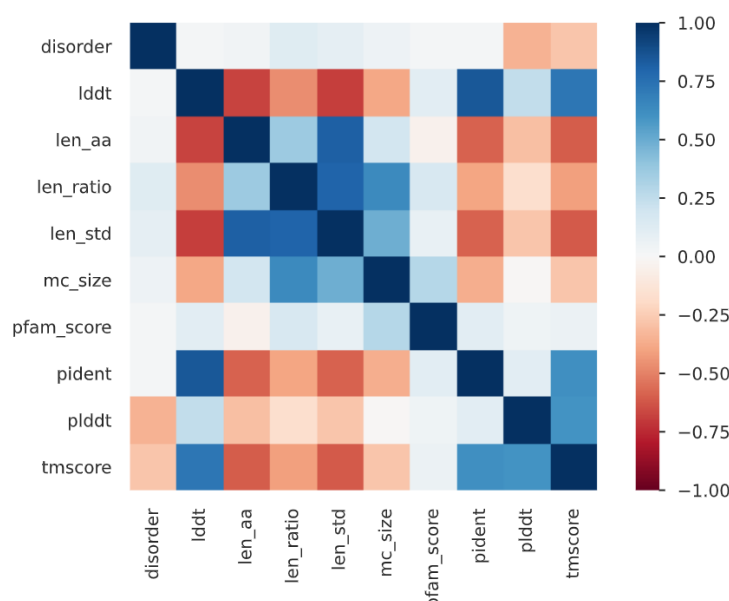


Figure 3.5: DPCstruct Pearson correlation matrix. Dark blue = strong positive correlation; dark red = strong negative correlation.

The four structural quality metrics: **tmscore**, **liddt**, **pident**, and **plddt** form a coherent positive cluster: metaclusters whose structures are confidently predicted by AlphaFold2 tend to show stronger internal structural consistency and higher intra-cluster sequence identity. The negative correlation between **disorder** and this quality block is expected: highly disordered regions are harder to align structurally and receive lower confidence scores by design. Crucially, **mc_size** is nearly uncorrelated with all quality metrics, confirming that a metacluster does not need to be large to be structurally coherent. This finding eliminated any need for a size-based filter in the DPCstruct views.

3.4.3 CATH and SCOP Structural Annotations

The CATH (1,274 rows) and SCOP (1,359 rows) annotation files each contain 15 columns describing the pairwise structural alignment between a DPCstruct representative and a reference fold from the corresponding database. The key alignment metrics are **qtm_score** (TM-score of the query), **alntmscore** (TM-score over the aligned region), **liddt**, **pident**, and **alnlen**.

Figure 3.6 plots TM-score against sequence identity for the full DPCstruct dataset and for each annotation file.

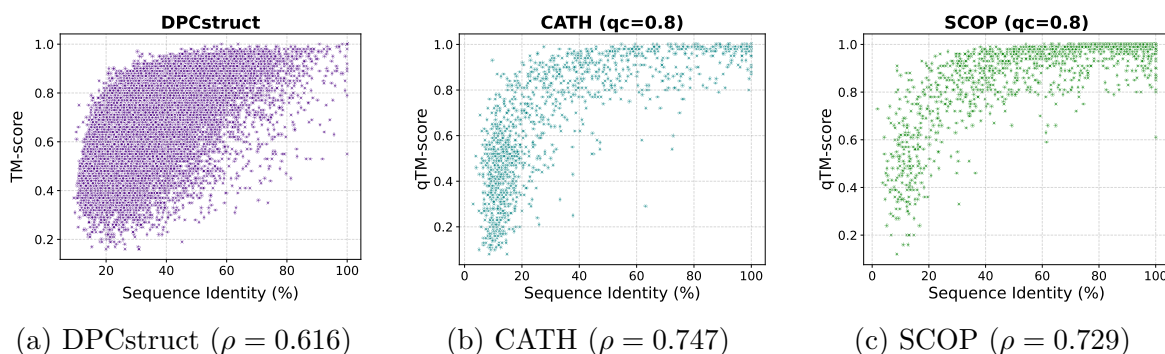


Figure 3.6: TM-score vs. sequence identity (**pident**) for DPCstruct overall (a), CATH annotations (b), and SCOP annotations (c). In all three panels, a dense cloud of points sits above TM-score = 0.5 but below 30% sequence identity, confirming that DPCstruct successfully detects remote structural homologs well inside the **twilight zone** [10].

The cloud of points with TM-score > 0.5 but sequence identity < 30% is the empirical signature of remote homology detection. DPCstruct groups these proteins together precisely because sequence-based methods cannot.

3.5 Key Findings and Their Impact on Design

The EDA step was not a formality: it produced four concrete decisions that shaped the final application.

1. **DPCfamB annotation fix.** The 18,989 inconsistent rows were corrected before any data touched the database, ensuring the annotation logic is coherent and trustworthy across all 81,384 DPCfam metaclusters.
2. **Three-way metacluster filter.** The steep drop in annotation quality from Standard (14.2% Equivalent) to DPCfamB (4.1% Equivalent) justified exposing all three dataset views: **DPCfam** / **Standard** / **DPCfamB** rather than merging everything silently behind one table.
3. **Dark matter must be visible.** With 40.8% of DPCfam and 49.3% of DPCstruct metaclusters carrying no Pfam label, hiding or deprioritising unannotated families would obscure the most scientifically interesting results. The application displays them prominently with UNKNOWN labels.
4. **No size-based filtering for DPCstruct.** The near-zero correlation between `mc_size` and structural quality confirmed that small metaclusters are as structurally reliable as large ones. No size threshold was applied, and all 28,246 metaclusters were loaded into the database.

3.6 From CSV to the Web Framework

The preprocessing stage produces two types of output, each handled differently in the application:

- **CSV files** [2] are loaded into PostgreSQL tables. They contain all queryable metadata: metacluster properties, protein-metacluster assignments, Pfam cross-references, and structural annotations.
- **Biological files** (FASTA, PDB, HMM, MSA) are stored as static assets and served directly to the user via download buttons on detail pages, without any database involvement.

Chapter 4 now picks up from here and describes how these cleaned CSV files were ingested into PostgreSQL, how the database schema was designed, and how Django's ORM exposes the data through the four supported query types.

Chapter 4

System Design and Architecture

This chapter provides the technical “*blueprint*” of our application. It details the choice of the **Model-View-Template** (MVT) architecture, the rationale behind the selected technology stack, the relational database design in **PostgreSQL**, and how these choices support the FAIR data principles.

4.1 Overview

The application is built with the **Django web framework** and follows Django’s **Model-View-Template** (MVT) architecture. In this pattern, the **model** layer manages data, the **view** layer handles request processing and response generation, and the **template** layer controls presentation. This separation of concerns improves maintainability and keeps the codebase easier to extend [30].

The full interaction cycle for a typical user request is illustrated in Figure 4.1.

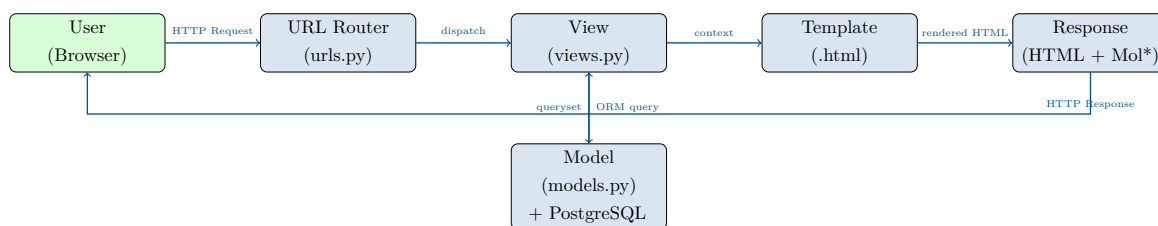


Figure 4.1: Simplified view of the Django MVT architecture. The user’s browser sends an HTTP request; the URL router dispatches it to the appropriate View. The View queries PostgreSQL through the ORM, receives a queryset, and passes it as context to the HTML template. The rendered page, which may include the Mol* 3D viewer for structural data, is then returned to the browser.

4.2 Technology Stack Justification

The choice of tools was driven by the specific needs of scientific data engineering:

- **Python:** Selected as the primary language because it dominates data science and bioinformatics, offering a rich ecosystem of libraries for parsing complex biological data formats.
- **Django** [30]: Chosen for its “**Don’t Repeat Yourself**” (DRY) philosophy and its ability to build maintainable, database-driven applications rapidly [31]. Its **Object-Relational Mapper (ORM)** allows us to interact with biological data through Python classes, avoiding error-prone raw SQL.
- **PostgreSQL** [32]: Selected over simpler engines like SQLite because it handles millions of records reliably and supports the advanced indexing strategies that our dataset sizes require. It integrates seamlessly with Django through the official `psycopg2` adapter.

4.3 Django Application Structure

To keep the codebase modular and maintainable, the platform is split into three specialized Django applications, each with its own models, views, and templates:

- **dpc:** The core application. It acts as the shared registry for all entities that both datasets rely on: UniProt protein records, Pfam domain identifiers, and the junction table that maps each UniRef50 protein to its Pfam domain annotations.
- **dpcfam:** Dedicated to the sequence-based clusters. It manages metacluster properties, the domain seed sequences linked to each metacluster, and the AlphaFold representative structures used for 3D visualization.
- **dpcstruct:** Dedicated to the structure-based clusters. It stores metacluster properties, the representative sequences, and the fold annotation tables that link metaclusters to entries in the CATH and SCOP databases.

4.4 Database Design

Each preprocessed **CSV file** [2] listed in Table 3.1 was mapped to a relational table in our PostgreSQL database. By defining the foreign key relationships between these entities, we established a schema organized into three functional zones.

4.4.1 Schema Overview

The PostgreSQL database contains ten tables organized into three interconnected zones, as illustrated in Figure 4.2. Think of our schema as three islands connected by bridges: the central **Core** island holds the shared entities that both datasets depend on, and the two outer islands hold the sequence-based and structure-based data respectively.

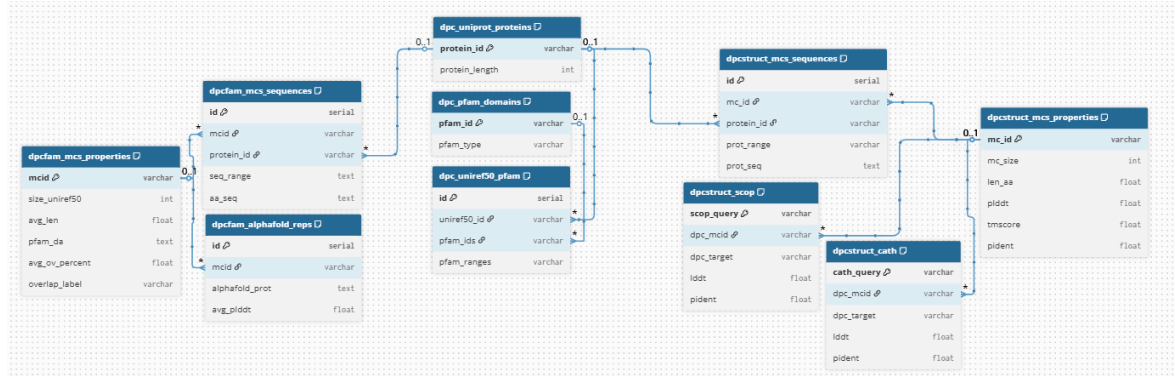


Figure 4.2: Entity-Relationship diagram of the database schema, generated using the <https://dbdiagram.io> platform. The architecture is organized into three functional zones: the **Core zone** (center) serves as the primary registry for both datasets, while the **Sequence zone** (left, DPCfam) and the **Structure zone** (right, DPCstruct) integrate with the core via `protein_id` foreign key relationships.

Zone 1 : Core (dpc)

The three core tables are the foundation of the whole schema. `dpc_uniprot_proteins` is the master protein registry: every protein that appears anywhere in the application must first exist here, identified by its UniProt accession (`protein_id`) and its total amino-acid length (`protein_length`). This single source of truth eliminates redundancy and makes it possible to draw a protein's full domain picture: Pfam, DPCfam, and DPCstruct matches from one query.

`dpc_pfam_domains` stores the universe of known Pfam family and clan identifiers (`pfam_id`) together with their type (Family or Clan). This table validates every Pfam search query at runtime: if a user types a Pfam ID that does not exist here, the application returns an informative error rather than an empty result.

`dpc_uniref50_pfam` is the junction table that connects proteins to their Pfam annotations. Each row records that a given protein carries a particular Pfam domain within a specific coordinate range. Because one protein can carry several non-overlapping domains, this table has a many-to-many character, hence the surrogate **BIGSERIAL** primary key. Foreign keys to both parent tables are enforced with **ON DELETE CASCADE**, so that removing a protein or a Pfam entry automatically cleans up all related rows.

Zone 2 : Sequence (**dpcfam**)

Three tables cover the sequence-based metaclusters. **dpcfam_mcs_properties** is the central summary table for DPCfam: each row is one metacluster identified by its **mcid** (e.g., **MC1042**), and its columns capture both the intrinsic properties of the cluster and its relationship with Pfam (see Table 3.2). **dpcfam_mcs_sequences** is the large mapping table that links every seed protein to its metacluster. Finally, **dpcfam_alphafold_reps** lists the reported AlphaFold2 representative structures for some metaclusters (38,668 MCs). This table feeds the “3D Viewer” tab on each DPCfam metacluster page.

Zone 3 : Structure (**dpcstruct**)

Four tables cover the structure-based metaclusters. **dpcstruct_mcs_properties** stores one row per structural metacluster: unlike the DPCfam equivalent, its columns emphasize 3D quality metrics (see Table 3.3). **dpcstruct_mcs_sequences** maps the representative protein structures to their metacluster. The two annotation tables, **dpcstruct_cath** and **dpcstruct_scop**, record the results of aligning DPCstruct metaclusters against CATH and SCOP folds. Each row links a specific CATH or SCOP fold identifier (the “query”) to a DPCstruct metacluster through the foreign key **dpc_mcid**, and stores the full set of alignment quality scores (query and target coverage, query/target/alignment TM-scores, IDDT, and percent identity).

4.4.2 Performance Optimization

To keep the application responsive when dealing with millions of rows, we implemented three indexing strategies:

B-Tree indexes on foreign key columns. By default, PostgreSQL does not automatically index foreign keys. Omitting these indexes would severely penalise every relational **JOIN** operation. To prevent this, every foreign key column in the mapping tables carries a standard B-Tree index. Specifically, **protein_id** in both **dpcfam_mcs_sequences** and **dpcstruct_mcs_sequences**, **mcid** in **dpcfam_mcs_sequences** and **dpcfam_alphafold_reps**, **mc_id** in **dpcstruct_mcs_sequences**, and **dpc_mcid** in both **dpcstruct_cath** and **dpcstruct_scop** are strictly indexed. With these indexes in place, cross-referencing a protein across all distinct datasets completes in milliseconds, regardless of table size.

GIN trigram indexes for Pfam text search. The **pfam_da** column in both **dpcfam_mcs_properties** and **dpcstruct_mcs_properties** is a free-text field containing concatenated Pfam IDs, separated by dashes (e.g., **PF00001-PF02985**). When a user searches for a specific Pfam ID, the application relies on a regular expression

to locate metaclusters containing that precise ID as an isolated token rather than a substring. Since standard B-Tree indexes cannot accelerate internal pattern matching, we enabled PostgreSQL's `pg_trgm` extension and applied a GIN (Generalized Inverted Index) to `pfam_da` across both tables. A GIN trigram index decomposes stored strings into overlapping three-character windows, indexing them simultaneously. During execution, the search pattern undergoes the same decomposition, allowing PostgreSQL to quickly intersect matching sets. This approach turns a costly full-table scan into an instantaneous index lookup.

Functional index for natural numeric sorting. Metacluster identifiers logically follow the pattern `MC{n}`, where n is an integer. While storing them as `VARCHAR` is structurally appropriate, a standard `ORDER BY mcid` sorts them lexicographically: `MC1`, `MC10`, `MC100`, `MC1000`, and eventually `MC2`. To enforce natural numerical sorting without burdening the application layer, we implemented a functional index that extracts the numeric component at the time of index creation (Figure 4.3).

```

61 -- Used to sort metaclusters numerically (MC1, MC2, MC10) instead of alphabetically.
62 CREATE INDEX IF NOT EXISTS idx_per_mcid_dpcfam ON dpcfam_mcs_properties (CAST(SUBSTRING(mcid FROM '[0-9]+') AS INTEGER));
93 -- Used to sort metaclusters numerically (MC1, MC2, MC10) instead of alphabetically.
94 CREATE INDEX IF NOT EXISTS idx_per_mcid_dpcstruct ON dpcstruct_mcs_properties (CAST(SUBSTRING(mc_id FROM '[0-9]+') AS INTEGER));

```

Figure 4.3: Functional index for natural numeric sorting of MCIDs.

The index uses a POSIX regular expression to isolate the digit sequence from the identifier string before converting it into a standard integer. Consequently, when Django processes queries, PostgreSQL directly utilizes this pre-built index to retrieve all rows in the correct numeric order instantly.

4.5 Query Types and Routing

To offer a single, unified entry point, all search traffic is routed through one central endpoint: `/search/?database={type}&query_id={id}`. The view behind this URL reads the two parameters and dispatches the request to the correct database lookup, as summarised in Table 4.1.

Table 4.1: Query types supported by the web application.

Query Type	Example Input	Result
DPCfam MCID	MC1042	Detail page for DPCfam metacluster 1042
DPCstruct MCID	MC5	Detail page for DPCstruct metacluster 5
Pfam ID	PF00001	All metaclusters linked to this Pfam family
UniProt ID	A0A182N2I3	Domain architecture page for this protein

This design means all database requests run through a unified search router. From the landing page, a researcher simply selects their target database (DPCfam, DPCstruct, Pfam, or UniProt) via a **radio button** and inputs their identifier. The application's routing view then handles the underlying complexity and dispatches to the correct logic. For DPCfam and DPCstruct MCIDs, the router performs an exact primary-key lookup and redirects directly to the detail page. For Pfam IDs, it queries the `pfam_da` column in both `dpcfam_mcs_properties` and `dpcstruct_mcs_properties` using indexed regular expressions, presenting the matching metaclusters from both datasets side by side. For UniProt IDs, it queries the `dpc_uniprot_proteins` table and efficiently fetches the related sequence mappings and `dpc_uniref50_pfam` domains via the application's ORM. It then assembles a unified domain architecture view that shows, on a single graphical scale, which DPCfam regions, DPCstruct regions, and Pfam domains cover which parts of the protein.

Table 4.2 provides a complete map of all URL patterns exposed by the application.

Table 4.2: All primary HTTP routes exposed by the Django web framework.

Endpoint (URL Pattern)	Method	Content & User Expectation
/	GET	Landing page with the universal search bar.
/search/?database={db}&query_id={id}	GET	Central router dispatching to the correct dataset or detail page.
/pfam/{pfam_id}/	GET	Metaclusters from DPCfam and DPCstruct that carry this Pfam domain.
/protein/{protein_id}/	GET	Graphical domain architecture for a single UniProt accession.
/dpcfam/mcs/	GET	Paginated, filterable table of all DPCfam metacluster properties.
/dpcfam/mcs/{mcid}/	GET	Detail page with four tabs: Summary, Sequences, 3D Viewer, Downloads.
/dpcstruct/mcs/	GET	Paginated table of DPCstruct metacluster properties, with CATH and SCOP sub-views.
/dpcstruct/mcs/{mc_id}/	GET	Detail page with four tabs: Summary, Sequences, 3D Viewer (Mol* 3D viewer), Downloads.
/dpcfam/downloads/ & /dpcstruct/downloads/	GET	Download portals for FASTA, HMM, MSA, and PDB files.
/about/ & /faqs/	GET	Static documentation and frequently asked questions.

4.6 REST API for Data Downloads

In its current state, each metacenter detail page provides direct download links to static biological files: FASTA seeds, HMM profiles, and MSA files for DPCfam, and FASTA sequences and zipped PDB files for DPCstruct. These files are served by **WhiteNoise**, a Python middleware that compresses and caches static assets efficiently without requiring a separate web server. In a planned future release, the global “Downloads” menu will be upgraded into a proper **REST API** using the **Django REST Framework (DRF)**. The planned endpoints will accept an **MCID** or a list of **MCIDs** and return the requested biological file in the appropriate format, making it straightforward for external scripts to programmatically fetch data. Filtering by sub-dataset (Standard DPCfam, DPCfamB, or DPCstruct) will also be supported.

With the architecture, the database schema, and the routing logic now in place, Chapter 5 moves from blueprint to construction. It shows how each of these design choices was turned into working code, from the landing-page search to the Mol* 3D viewer embedded in every DPCstruct metacenter page.

Chapter 5

Implementation

In chapter 4, we presented the blueprint of our system. This chapter now walks through how that blueprint was actually built from the home page to the 3D molecular viewer embedded inside each DPCstruct metacluster page. We follow the same four-query logic introduced earlier, explaining each part of the user interface.

5.1 Technology Stack

Our application is written entirely in **Python** and served by **Django 6.0.1** [30]. The key packages are listed in Table 5.1.

Table 5.1: Python dependencies (`requirements.txt`).

Package	Role
Django 6.0.1	Web framework: URL routing, ORM, templating
psycopg2-binary 2.9.11	PostgreSQL adapter used by Django's ORM
django-tables2 2.8.0	Renders querysets as paginated, sortable HTML tables
django-filter 25.2	Attaches filter forms to querysets
whitenoise 6.11.0	Serves static files (CSS, JS, PDB, FASTA) efficiently
sqlparse 0.5.5	SQL query formatting used internally by Django
python-decouple 3.8	Reads configuration and secrets (database credentials, the admin flag) from the <code>.env</code> file

On the frontend, the application relies on **Bootstrap 5.3.2** loaded from a CDN, which handles all responsive layout and navigation components. No custom JavaScript framework (React, Vue, etc.) was introduced; we relied simply on plain JavaScript.

5.2 The Landing Page and Search

The application has a single entry point: the landing page at /. From there, the user selects a database source (**DPCfam**, **DPCstruct**, **Pfam**, or **UniProt**) using **radio buttons** and types a valid identifier into a text field. Submitting the form sends a **GET** request to `/search/?database={type}&query_id={id}`.

The central `search()` view in `dpc_fam_and_struct_webapp/views.py` handles routing. Its logic is to check whether the submitted identifier exists in the relevant database table and, if so, redirects the user directly to the correct detail page. If not, it attaches an error message to the response and returns the user to the landing page.

5.3 DPCfam Metacluster Pages

The DPCfam section of the application is served by the `dpcfam` Django app and exposes two main views.

5.3.1 The Metacluster List

The list view at `/dpcfam/mcs/` is powered by `DpcfamMcsPropertyListView`, which combines `SingleTableMixin` (from `django-tables2`) with `FilterView` (from `django-filter`). The queryset is sorted numerically, not alphabetically, by casting the integer part of the MCID to a PostgreSQL integer (Figure 5.1):

```

32     qs = DpcfamMcsProperty.objects.annotate(
33         mc_num=RawSQL(
34             "CAST(SUBSTRING(mcid FROM '[0-9]+') AS INTEGER)",
35             [],
36             output_field=IntegerField()
37         )
38     ).order_by('mc_num')

```

Figure 5.1: Numeric sorting of DPCfam MCIDs.

The page also exposes a **three-way dataset filter**: a user can browse **DPCfam** (all 81,384 metaclusters), **Standard DPCfam** (those with ≥ 50 seeds), or **DPCfamB** (those with 25 - 49 seeds), using toggle buttons at the top of the page. This design decision was directly motivated by the EDA findings in Chapter 3, which showed a sharp quality gap between the two subsets.

5.3.2 The Metacluster Detail

Navigating to `/dpcfam/mcs/{mcid}/` opens the detail page for a single metacluster. The page uses a **Bootstrap tab navigation bar** to organise four panels:

- (1) **Summary** : Table listing all properties of the metacluster (size, average length, Pfam dominant architecture, overlap label, etc.). Each Pfam ID in the `pfam_da` field is rendered as a hyperlink that redirects to the Pfam query results page.
- (2) **Sequences** : Paginated table (10 rows per page) listing every seed sequence in the metacluster with its UniProt ID, domain coordinate range, sequence length, and the first 85 characters of the amino acid sequence. Each UniProt ID is also a hyperlink to the protein detail page.
- (3) **3D Viewer** : AlphaFold representative structure (if any) for the metacluster, served through a table of clickable link to the [AlphaFoldDB](#). This is different from DPCstruct, where structures are rendered inline, as explained in Section 5.5.
- (4) **Downloads** : Direct download links for the seed FASTA file, the Multiple Sequence Alignment (MSA), and the Profile Hidden Markov Model (HMM) of the metacluster.

The Django view builds the context by querying two managers: `self.object.sequences` for the seed table and `self.object.alphafolds` for the AlphaFold representative entries. File paths for downloads are constructed dynamically using Django's `static()` utility, pointing into the `production_files/dpcfam/` subtree of the static directory.

5.4 DPCstruct Metacluster Pages

The DPCstruct section mirrors the DPCfam section in structure but reflects the additional richness of the structural dataset: three different views of the data are available in the list page, and the detail page includes an inline 3D molecular viewer.

5.4.1 The Metacluster List

The list view at `/dpcstruct/mcs/` is handled by `DpcStructMetaclustersListView`, a `TemplateView` subclass that manages three sub-views toggled by a URL parameter (`?view=properties|cath|scop`):

- **Properties**: the main table of structural metacluster characteristics.
- **CATH**: alignment data between DPCstruct representatives and CATH folds.
- **SCOP**: equivalent data for SCOP folds.

For the Properties view, the same numeric MCID sorting trick used in DPCfam is applied here. Each view also supports an optional search by MCID and, for CATH and SCOP, by fold identifier. All three views paginate at ten rows per page using `django-tables2`'s built-in `paginate()` method.

5.4.2 The Metacluster Detail

The detail page at `/dpcstruct/mcs/{mc_id}/` is structured around the same four-tab layout as DPCfam (Summary, Sequences, 3D Viewer, Downloads). What makes it different is the **3D Viewer** tab: instead of linking out to an external resource, it renders representative PDB structures directly inside the browser, as described below.

5.5 Mol* Integration

Each DPCstruct metacluster has at most two representative PDB files: the **cluster center** and the **member with the highest pLDDT score** [10]. In some clusters, they are the same, resulting in one representative (e.g, **MC2011**). These files are AlphaFold2 predictions, stored under `static/production_files/dpcstruct/dpcstruct_reps_pdb/{mc_id}_pdb/`. The goal is to let the user interact with these structures directly in the browser, without installing any external software.

5.5.1 Mol* in a nutshell

Mol* [16] is a modern, open-source JavaScript library for the 3D visualization of macromolecular structures in the web browser. It was jointly developed by **PDBe** (Protein Data Bank in Europe, EMBL-EBI) and **RCSB PDB** (Research Collaboratory for Structural Bioinformatics), and serves as their primary 3D viewer. It is:

- **Client-side:** It runs entirely in the browser using **WebGL** (Web Graphics Library), the same technology that powers browser-based 3D games, so no server-side GPU is needed.
- **Feature-rich:** It allows users to rotate, zoom, measure distances, color by property, take screenshots, switch representations, and view the sequence.
- **Open-source:** It is released under the MIT license and is actively maintained.
- **Embeddable:** It can be loaded from a CDN¹ using only two files (one CSS and one JS) and initialized with a few lines of JavaScript.

¹A Content Delivery Network (CDN) is a system of distributed servers that cache and deliver files quickly based on the user's location.

5.5.2 Why PDBe-Molstar

Our initial implementation used **vanilla Mol*** [16]. It worked, but it has no built-in awareness of AlphaFold structures: by default, it colours the structure by atom type, which is not very informative for predicted models. We therefore switched to **PDBe-Molstar** (`pdbe-molstar`), a pre-configured wrapper around Mol* maintained by the Protein Data Bank in Europe. PDBe-Molstar adds one key feature that was decisive for our use case: it recognises AlphaFold PDB files and automatically applies the **AlphaFold confidence colouring scheme**, where each residue is coloured by its per-residue pLDDT score. This gives researchers an immediate visual sense of which parts of a structure are reliably predicted and which are uncertain. We present the pLDDT colour scale in Table 5.2.

Table 5.2: AlphaFold pLDDT confidence colour scheme as rendered by PDBe-Molstar.

pLDDT range	Interpretation	Colour
> 90	Very high confidence	Dark blue
70 – 90	Confident	Light blue / cyan
50 – 70	Low confidence	Yellow
< 50	Very low confidence	Orange

A custom overlay legend at the bottom-left of the canvas keeps this colour scale visible at all times, without opening any menu inside the viewer.

5.5.3 Loading PDBe-Molstar

Like the original Mol*, PDBe-Molstar is loaded from a CDN (we used [jsDelivr](#)) using two files: a CSS stylesheet and a JavaScript bundle:

```
7 <!-- PDBe Molstar Viewer CSS -->
8 <link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/pdbe-molstar@3.11.0/build/pdbe-molstar-light.css">
```

(a) CSS.

```
268 <!-- PDBe Molstar Viewer JS -->
269 <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/pdbe-molstar@3.11.0/build/pdbe-molstar-plugin.js"></script>
```

(b) JS script.

Figure 5.2: Loading PDBe-Molstar from a CDN in `dpcstruct_detail.html` template. No local installation is needed; the browser caches the files after the first load.

5.5.4 The Django Backend: Preparing PDB File Paths

The Django view `DpcStructDetailView` is responsible for locating the PDB files on disk and passing their URLs to the template. Inside `get_context_data()`, the view

scans the per-metaccluster PDB folder using Python's `os.listdir()`, builds a list of dictionaries (each containing a `name` and a `url`), and places it in the template context under the key `pdb_files` (Figure 5.3):

```

135 # 3D Viewer: list PDB files in this MC's folder
136 pdb_folder = os.path.join(
137     settings.BASE_DIR, 'static',
138     'production_files', 'dpcstruct', 'dpcstruct_reps_pdb',
139     f'{mc_id}_pdb'
140 )
141 pdb_files = []
142 if os.path.isdir(pdb_folder):
143     for fname in sorted(os.listdir(pdb_folder)):
144         if fname.endswith('.pdb'):
145             pdb_files.append({
146                 'name': fname,
147                 'url': static(
148                     f"production_files/dpcstruct/dpcstruct_reps_pdb/"
149                     f"{mc_id}_pdb/{fname}"
150                 ),
151             })
152 context['pdb_files'] = pdb_files
  
```

Figure 5.3: Building the `pdb_files` context variable in `DpcStructDetailView`.

5.5.5 Passing Data to JavaScript

The template cannot call Python functions directly. To transfer the `pdb_files` list to JavaScript, we use Django's built-in `json_script` template filter, which serialises a Python object to JSON and embeds it inside a `<script>` block with a known `id` attribute (Figure 5.4):

```

234 <!-- Pass PDB URLs to JavaScript -->
235 {{ pdb_files|json_script:"pdb-files-data" }}
  
```

Figure 5.4: Injecting `pdb_files` into the HTML safely.

On JavaScript side, the data is recovered with a standard `JSON.parse` call (Figure 5.5):

```

292 // --- Mol* 3D Viewer ---
293 var dataEl = document.getElementById('pdb-files-data');
294 if (!dataEl) return;
295 var PDB_FILES = JSON.parse(dataEl.textContent);
296 if (!PDB_FILES || PDB_FILES.length === 0) return;
  
```

Figure 5.5: Reading the JSON data in JavaScript.

5.5.6 Lazy Initialisation and Structure Switching

Because 3D rendering is computationally intensive, the viewer is **not initialised** when the page loads. Instead, we listen for the Bootstrap `shown.bs.tab` event on the “3D

Viewer” tab button. Only when the user explicitly clicks that tab does the initialisation function run for the first time (Figure 5.6):

```

301 // Initialize Mol* when the 3D Viewer tab is first shown
302 var viewerTab = document.getElementById('viewer-tab');
303 viewerTab.addEventListener('shown.bs.tab', function () {
304     if (!viewerInitialized) {
305         viewerInitialized = true;
306         initView(PDB_FILES[0].url);
307     }
308 });
  
```

Figure 5.6: Lazy loading: the viewer is initialised only when the tab is opened.

The `initViewer()` function instantiates a new `PDBeMolstarPlugin` object, passes it a configuration object, and calls `render()` on the HTML `<div>` container (Figure 5.7):

```

320 async function initView(pdbUrl) {
321     viewerInstance = new PDBeMolstarPlugin();
322     var options = {
323         customData: {
324             url: pdbUrl,
325             format: 'pdb'
326         },
327         alphafoldView: true,
328         bgColor: {r:255,g:255,b:255},
329         hideControls: false, /* Show all controls except those explicitly hidden below */
330         sequencePanel: true, /* Show sequence panel at the top */
331         leftPanel: true, /* Show left panel with structure details */
332         landscape: true, /* Forces landscape layout, making left panel visible without going full screen
333         hideCanvasControls: ['selection', 'animation']
334     };
335     viewerInstance.render(document.getElementById('molstar-viewer'), options);
336 }
  
```

Figure 5.7: Initialising the PDBe-Molstar viewer with AlphaFold coloring.

The `alphafoldView: true` option is the key setting that activates the pLDDT-based colouring described in Table 5.2. The `sequencePanel` and `leftPanel` options keep the residue sequence bar and the structure control panel visible by default. The `landscape` flag forces a horizontal layout, which fits the page without requiring full-screen mode.

When the user clicks the second structure button (if a second PDB is available), the viewer is updated by calling `render()` again with the new URL, effectively replacing the previous structure (Figure 5.8):

```

338 // Handle PDB selector buttons
339 document.querySelectorAll('.pdb-selector').forEach(function (btn) {
340   btn.addEventListener('click', async function () {
341     // 1. Remove "active" color from all buttons, add to the clicked one
342     document.querySelectorAll('.pdb-selector').forEach(function (b) {
343       b.classList.remove('active');
344     });
345     this.classList.add('active');
346     // 2. Extract the URL stored on the button
347     var url = this.getAttribute('data-pdb-url');
348     // 3. Update the 3D Viewer
349     if (viewerInstance) {
350       var options = {
351         customData: {
352           url: url,
353           format: 'pdb'
354         },
355         alphafoldView: true,
356         bgColor: {r:255,g:255,b:255},
357         hideControls: false,
358         sequencePanel: true,
359         leftPanel: true,
360         landscape: true,
361         hideCanvasControls: ['selection', 'animation']
362       };
363       viewerInstance.render(document.getElementById('molstar-viewer'), options);
364     }
365   });
366 });
367
368

```

Figure 5.8: Switching between the two representative structures.

5.6 The Pfam Query

When a user submits a Pfam ID (e.g., PF00001) through the landing page search form, the `search()` router first checks that the ID exists in the `dpc_pfam_domains` table and then redirects to `/pfam/{pfam_id}/`, served by the `pfam_detail()` view.

The view queries both the DPCfam and DPCstruct metacluster property tables for any row whose `pfam_da` field contains the given Pfam ID. Because `pfam_da` is a text column storing dash-separated identifiers (e.g., PF00001-PF00456), a simple equality filter would miss multi-domain architectures. We therefore use a PostgreSQL regex filter (Figure 5.9):

```

64 # Get DPCfam metaclusters with this Pfam domain
65 dpcfam_metaclusters = DpcfamMcsProperty.objects.filter(
66   pfam_da__regex=rf'(^|-){pfam_id}(-|$)'

```

Figure 5.9: Regex filter for Pfam ID in multi-domain architectures.

This pattern matches the Pfam ID only at the start or end of the string, or surrounded by dashes, ensuring that PF00001 does not match PF000012 by accident. This query is accelerated by the GIN trigram index created on `pfam_da` during database initialisation (see subsection 4.4.2).

The results are presented in a two-column layout: the left column shows the DPCfam metaclusters and their average Pfam overlap percentage, while the right column shows

the DPCstruct metaclusters with their global Pfam consistency score. Each percentage is visualised as a “battery” bar that changes colour from red to green depending on the overlap/score value, giving the reader an instant qualitative signal of how well the cluster matches the queried family. Then, special handling is included for Pfam clan identifiers, which are redirected to a different InterPro URL than regular family entries.

5.7 The UniProt Query

Submitting a UniProt accession (e.g., P05067) routes the user to `/protein/{protein_id}/`, served by the `protein_detail()` view. This page is the most visually rich in our application: it presents a **domain architecture diagram** that maps all DPCfam, DPCstruct, and Pfam annotations simultaneously onto the full-length protein sequence.

5.7.1 The Domain Architecture Diagram

The diagram is built using only HTML and CSS absolute positioning; no external charting library was needed. The view computes, for each protein domain, a left offset and a width as percentages of the full protein length (Figure 5.10):

```

154     'width': ((end - start) / protein.protein_length) * 100 if protein.protein_length else 0,
155     'left': (start / protein.protein_length) * 100 if protein.protein_length else 0
  
```

Figure 5.10: Computing domain positions as percentages of protein length.

Three horizontal *tracks* are drawn: two above the scale line for both DPCfam domains (green palette) and DPCstruct domains (red palette), and one below for Pfam domains (blue palette). Hovering over any domain box triggers a CSS tooltip showing the exact identifier and coordinate range.

5.7.2 Toggles and Responsive Layout

Below the diagram, three checkboxes allow the user to show or hide each track independently. Toggling a checkbox fires a plain JavaScript function that adjusts both the diagram track opacity and the column width of the corresponding detail table below the diagram. When only one track is visible, that table automatically expands to fill the full width of the page.

5.8 The Administration Panel

By default and by construction, the Django administration site at `/admin/` is **read-only** for all models: an administrator can inspect every table but cannot add, change, or delete

any row, not even a superuser. To enable full CRUD, set `DPCEXPLORER_ADMIN_WRITABLE=True` in the `.env` file and restart the server; permissions then follow Django's normal per-user rules. We keep read-only as the default because the database is generated by a reproducible pipeline, which stays the single source of truth. For the full design, including the performance tuning that keeps the large tables responsive in the admin, we refer the reader to the repository's `ADMIN_PANEL.md` [1].

Chapter 6

Results and Discussion

This chapter walks through the final web application from the user’s perspective. We focus on how the tool looks and what it delivers. We also evaluate how well the system aligns with the FAIR principles introduced in Chapter 2 and discuss its reproducibility.

6.1 Application Demonstration

The application is currently running locally at <http://127.0.0.1:8000/> and will soon be publicly accessible at <https://dpcexplorer.areasciencepark.it/>, where the IT team at **Area Science Park** will deploy it using **Docker** containers orchestrated by **Kubernetes** on the **ORFEO HPC cluster**. Key pages are:

6.1.1 Landing Page

The first thing a user sees is the landing page, shown in Figure 6.1. It is deliberately minimal: a short welcome message, a set of four radio buttons to select the target database (**DPCfam**, **DPCstruct**, **Pfam**, or **UniProt**), and a single text field for the query identifier. Pressing *Search* triggers the central routing logic described in Chapter 4, which validates the ID and redirects the user to the appropriate detail page.

Figure 6.1: The unified landing page. Users select a database source and type a valid identifier to reach any metacluster, Pfam family, or UniProt protein in the system.

6.1.2 DPCfam Metacluster Table

Clicking on *Metaclusters/DPCfam* in the navigation bar, or choosing `/dpcfam/mcs/`, opens the metacluster table shown in Figure 6.2. By default, all 81,384 metaclusters are listed and paginated, ten rows per page, sorted numerically (MC1, MC3, ...). Three toggle buttons at the top right let the user restrict the view to **Standard DPCfam** (size ≥ 50), **DPCfamB** (size 25 - 49), or keep the full **DPCfam** combined set.

MCID	Size UniRef50	Avg. Len.	Std. avg. len	% LC	% CC	% DIS	Avg. TM	Pfam DA	% DA	% Avg. Ov.	Overlap Label
MC1	17931	185.68	28.77	4.72	0.0	18.44	0.01	PF13614	44.23	80.82	Equivalent
MC3	28	215.14	25.44	0.05	0.38	30.15	0.0	PF14631	85.71	24.57	Shifted
MC4	617	59.91	6.07	4.99	0.0	1.87	1.26	PF03600	62.84	7.54	Shifted
MC5	26	253.19	23.11	0.05	0.0	7.51	0.0	PF12937	68.0	16.89	Extended
MC7	32	65.53	3.3	0.03	0.0	0.1	0.0	PF01243	74.07	6.27	Shifted
MC11	32	134.97	11.99	0.02	0.0	15.61	0.0	UNKNOWN	0.0	0.0	NONE
MC15	139	81.21	5.05	4.96	0.18	13.81	0.03	UNKNOWN	0.0	0.0	NONE
MC17	26	243.69	25.93	0.03	0.0	31.88	0.0	PF08590	90.0	5.31	Shifted
MC18	41	143.29	8.57	0.04	0.0	1.12	0.13	PF18736	96.43	94.74	Equivalent
MC19	120	71.57	7.7	8.85	2.25	4.86	1.69	PF11915	94.07	13.66	Shifted

Figure 6.2: The DPCfam metacluster table with the default view (**DPCfam**, 81,384 rows). The three toggle buttons on the top right switch between DPCfam, Standard, and DPCfamB. The search bar on the top left allows direct lookup by MCID. Columns include Metacluster intrinsic properties, and their Pfam Annotations (see Table 3.2).

Each MCID in the table is a clickable link. It leads to that metacluster's own detail page, where the full set of properties, seed sequences, AlphaFold representatives, and downloadable files can be explored. Figure 6.3 shows the detail page for one metacluster. It is organized into four tabs: *Summary*, *Sequences*, *3D Viewer*, and *Downloads*. The Summary tab presents the key metrics from the `dpcfam_mcs_properties` table, simply the MCID row, as illustrated above in Figure 6.2. The Sequences tab lists all seed

domains with their UniProt ID, range, and amino acid sequence, paginated 10 rows at a time. The 3D Viewer tab shows the AlphaFold-predicted structure of the cluster's representative (if any) retrieved from the AlphaFoldDB, and the Downloads tab offers one-click access to the FASTA seeds, MSA profile, and HMM model for that metacluster.

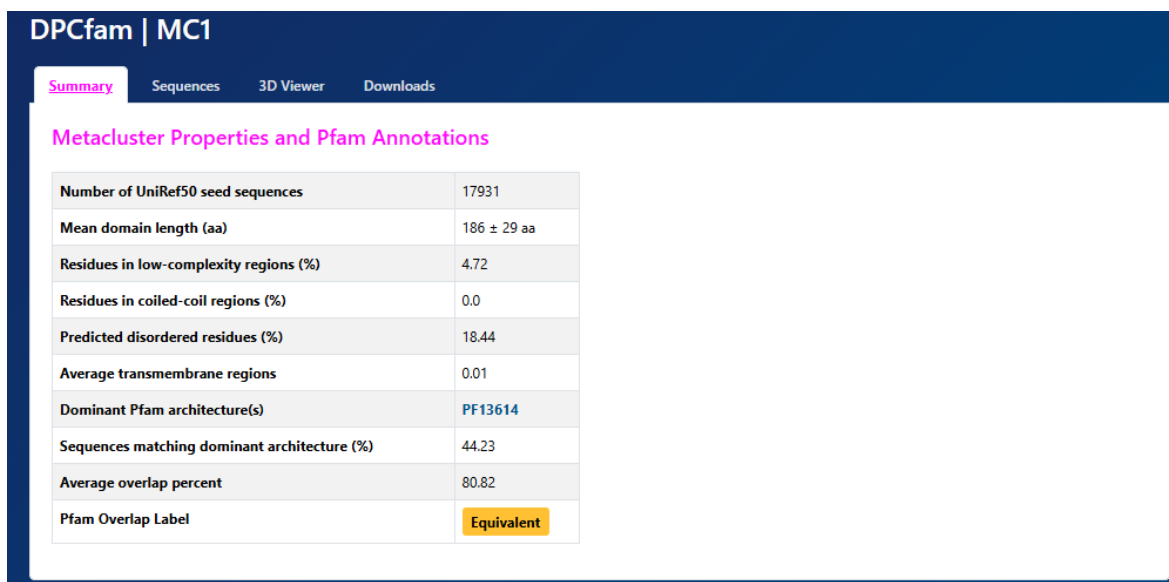


Figure 6.3: Detail page for DPCfam MC1 (Summary tab active). The table lists MC1 biological properties, including its dominant Pfam architecture rendered as a clickable hyperlink, the percentage of seeds matching it, and the qualitative overlap label.

6.1.3 DPCstruct Metacluster and 3D Viewer

The DPCstruct section follows the same layout philosophy, but it handles structural data. Navigating to `/dpcstruct/mcs/` opens the paginated table shown in Figure 6.4. Three view buttons at the top right switch between the **Properties** tab (the default), the **CATH** annotations tab, and the **SCOP** annotations tab.

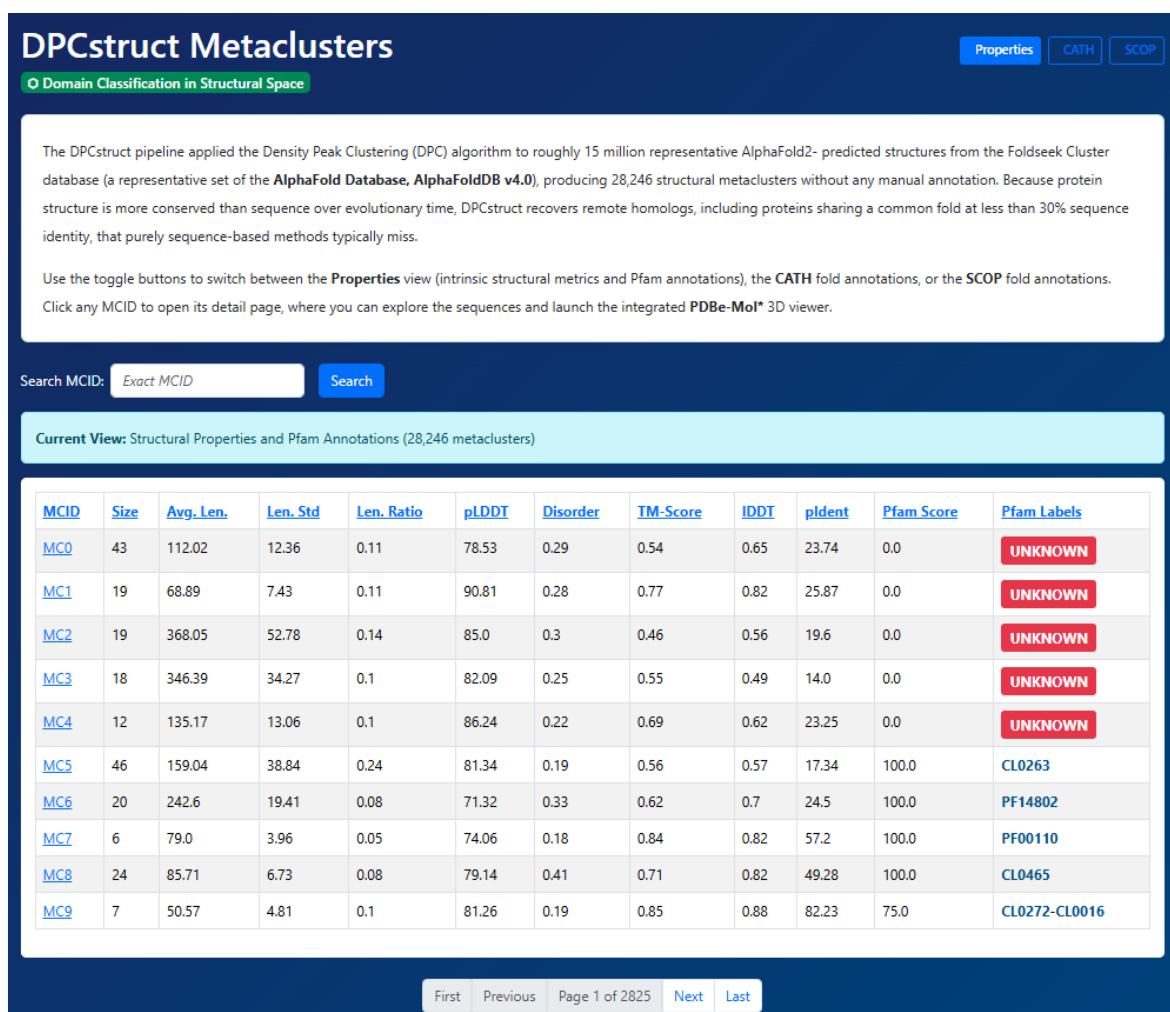


Figure 6.4: The DPCstruct metacluster table in the default Properties view (28,246 rows). The three toggle buttons at the top right switch between the Properties, CATH, and SCOP annotation views. Columns include structural quality metrics such as average pLDDT, TM-score, IDDT, and Pfam Annotations (see Table 3.3).

Clicking any metacluster opens its detail page. The most distinctive feature here is the *3D Viewer* tab, powered by the Mol* library introduced in Chapter 5. Figure 6.5 shows a representative PDB structure loaded inside the browser without any plugin or external software. The structure is colored by AlphaFold2 pLDDT confidence, and the viewer allows rotation, zoom, distance measurements, and representation changes in real time. A custom legend overlay (bottom-left of the canvas) explains the color scale.

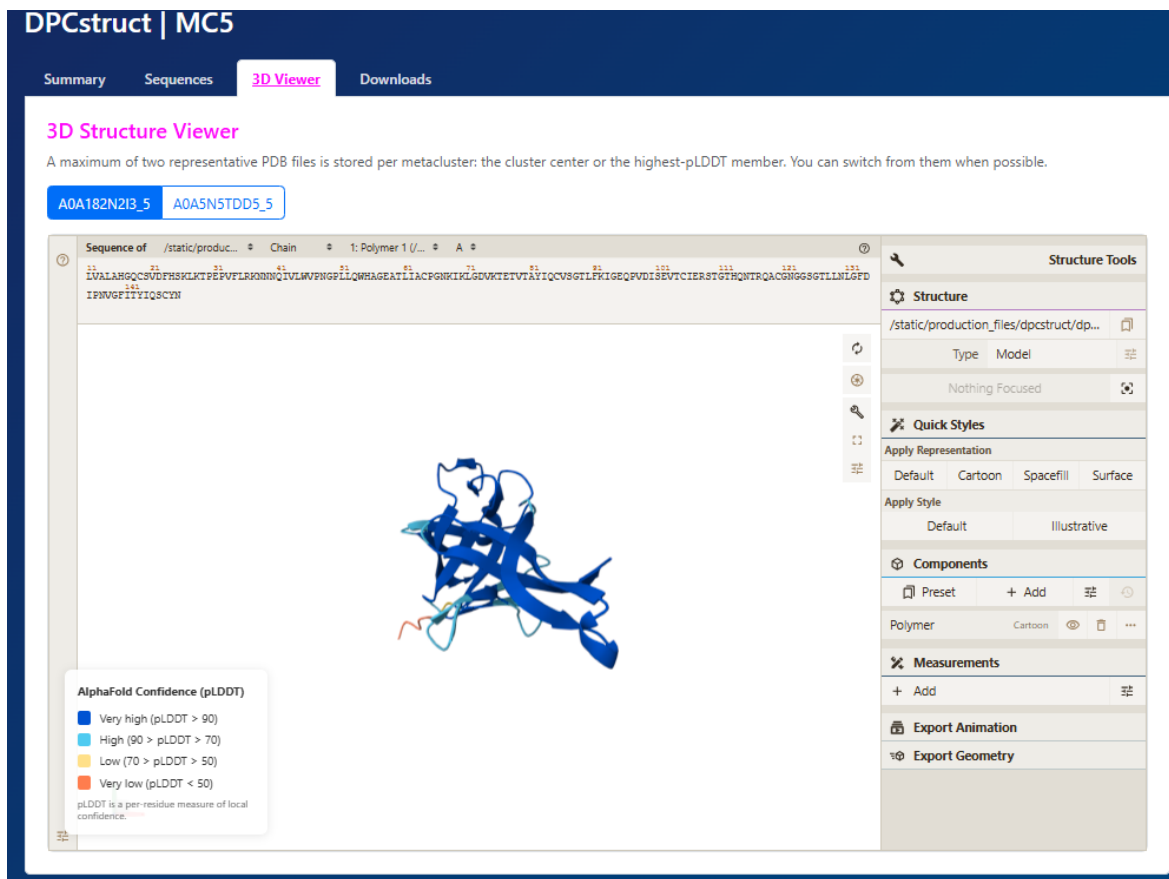


Figure 6.5: The integrated Mol* 3D viewer on the detail page of DPCstruct MC5 (3D Viewer tab active). The structure is colored by AlphaFold2 pLDDT confidence: dark blue indicates very high confidence ($\text{pLDDT} > 90$), light blue indicates high confidence (70–90), yellow indicates low confidence (50–70), and orange indicates very low confidence (< 50). The viewer runs entirely in the browser using **WebGL**; no server-side GPU is required.

An important design detail: Mol* is loaded *lazily* (see subsection 5.5.6). It is only initialized when the user explicitly clicks the 3D Viewer tab. This keeps the page responsive for users who only need the sequence data, and avoids loading a heavyweight WebGL canvas for no reason.

6.1.4 Pfam Query Results

When a user enters a valid Pfam identifier (e.g. PF02990) and selects **Pfam** from the landing page, the application queries both the DPCfam and DPCstruct tables and renders the results side by side, as shown in Figure 6.6. For DPCfam, metaclusters are listed by their average Pfam overlap percentage, displayed as a battery-style progress bar. For DPCstruct, they are listed by their Pfam (global)consistency score, with the associated Pfam families or clans rendered as clickable badges.

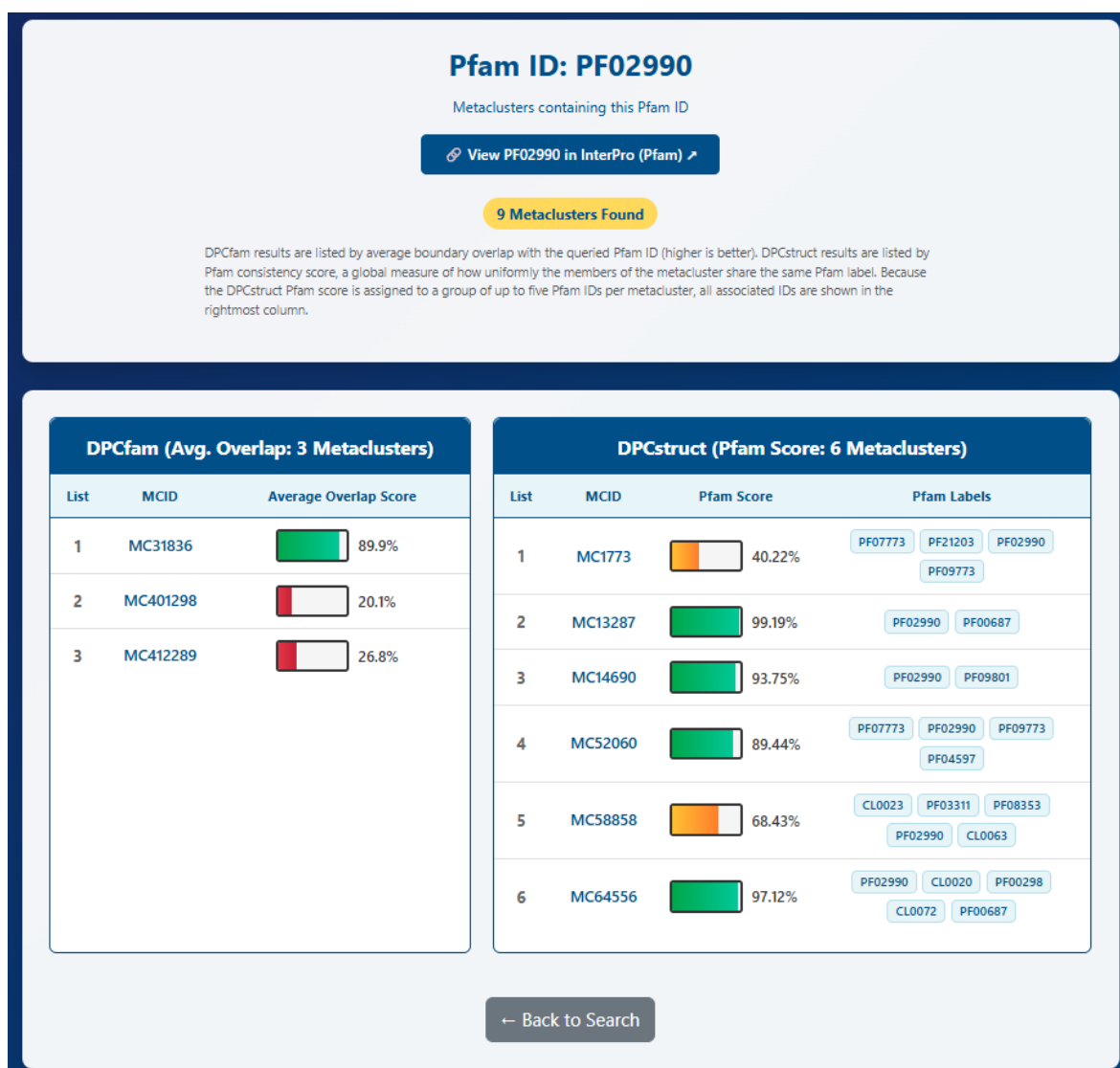


Figure 6.6: Results for a Pfam query. The page is divided into two columns: DPCfam metaclusters (left) listed by average overlap percentage, and DPCstruct metaclusters (right) listed by Pfam (global)consistency score. Each metacluster ID is a direct link to its detail page. The battery-style bars give an immediate visual sense of how strongly each metacluster overlaps with the queried Pfam family.

It is worth noting that the `pfam_score` field in the DPCstruct database is a global score assigned to a group of up to five Pfam IDs per metacluster, rather than to each Pfam ID individually. Therefore, where applicable, we display the full list of Pfam IDs to which the global score is assigned and from which the displayed ID is selected.

6.1.5 UniProt Query Results

The UniProt query is the most visual feature of our application. Given a valid UniProt accession (e.g. A0A182N2I3), the system retrieves the full protein record from the `dpc_uniprot_proteins` table, identifies all its DPCfam domains, DPCstruct domains,

and Pfam annotations, and renders the domain architecture diagram shown in Figure 6.7.

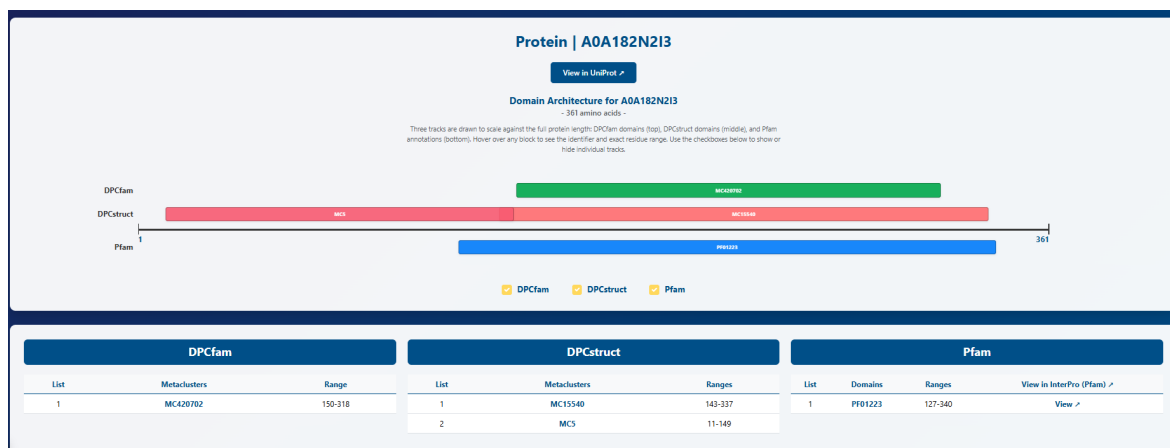


Figure 6.7: Domain architecture diagram for UniProt A0A182N2I3. Three horizontal tracks are drawn above and below a scale line representing the full protein length: DPCfam domains (top), DPCstruct domains (middle), and Pfam annotations (bottom). Each colored block is a detected domain region, with its position on the sequence determined by the range stored in the database. The three toggle checkboxes below the diagram allow users to show or hide each track independently. Below the diagram, a table lists all detected domains with their metacluster IDs and coordinate ranges.

The three tracks are drawn in proportion to the protein’s full length, using percentage-based CSS `left` and `width` values computed server-side in the `protein_detail` view (see subsection 5.7.1). Hovering over any domain block reveals a tooltip with the MCID and exact residue range. Domains are sorted by width in descending order before rendering, so that smaller domains are drawn on top of larger ones and are not hidden.

6.2 System Evaluation

The indexing strategy of Chapter 4 promised fast lookups regardless of table size. This section checks that promise against reality.

We ran a verification suite, `verify_dpcexplorer_db_indexes.sql` [1], on a fully populated local instance (Ubuntu 24.04.3 LTS, PostgreSQL 16.11, a 4-core virtual processor, 16 GB RAM). Using PostgreSQL’s `EXPLAIN (ANALYZE, BUFFERS)`, it confirms that each index is built over real data and that the planner actually chooses it for the query it was designed for: the GIN trigram index for Pfam text search, the functional index for natural MCID sorting, and the B-Tree foreign-key indexes for the protein and metacluster joins. It also reports the live usage counters and the index-usage rate per table. The complete output is archived in the repository report `dpcexplorer_schema_outputs.txt` [1].

Every audited query pattern resolved through an **index scan** rather than a sequential

scan, exactly the behaviour the design intended, and each one returns in **under 5 ms**. The live counters tell the same story: the four most-queried tables alone had already been used by the query planner more than **65 million times**. Table 6.1 reports the index-usage rate per table.

Table 6.1: Index usage rate per table after real application traffic, from Section 7 of the verification report. The four most-queried tables all reach 100%.

Table	Live rows	Usage rate
dpc_uniprot_proteins	14,121,433	100%
dpcfam_mcs_sequences	16,602,634	98.2%
dpc_uniref50_pfam	14,554,934	83.3%
dpcstruct_mcs_sequences	1,642,061	97.3%
dpcfam_mcs_properties	81,384	100%
dpcfam_alphafold_reps	38,668	97.0%
dpcstruct_mcs_properties	28,246	100%
dpc_pfam_domains	19,352	100%
dpcstruct_scop	1,359	84.9%
dpcstruct_cath	1,274	83.3%

The four **most-queried** tables, on which every user-visible query ultimately depends, all reach a 100% indexed-access rate. The lower rates on the remaining tables are a counting artefact: PostgreSQL’s counters include the one-time sequential **COPY** operations that bulk-loaded the data during initialisation. As real user traffic accumulates, those rates climb toward 100% as well.

6.3 FAIR4RS Assessment

While the upstream datasets [12, 13] and our preprocessed deposit [2] were already aligned with the original 2016 **FAIR data principles** [14] through their Zenodo deposits, here we focus on **DPCexplorer as a software artifact** and evaluate it against the **FAIR Principles for Research Software (FAIR4RS)** [28]. Unlike data, research software is executable, constantly evolves, and often functions as a complex collection of interdependent parts. FAIR4RS therefore extends the original four principles with software-specific sub-criteria, which we address below.

Findable: persistent identification and metadata. To make the project easy for both humans and machines to find, we assigned globally unique and persistent identifiers to the work (**F1**). The software application has its own citable DOI [1], while the preprocessed datasets are stored under a separate DOI [2] (**F1.1**). Specific releases, such as version v1.0.3, are assigned their own distinct records under a single concept DOI (<https://doi.org/10.5281/zenodo.20575268>) that always resolves to the latest release (**F1.2**). The project is described with rich, machine-readable metadata through a `CITATION.cff` file and an About page (**F2**), which explicitly include the project's identifiers to keep a clear link to the software (**F3**). At the metacluster level, every entry has a stable, bookmarkable URL (e.g., `/dpcfam/mcs/MC1/`), and the universal search bar resolves any known biological identifier directly. Finally, by registering all artifacts on Zenodo, the metadata is searchable and indexable in a global repository (**F4**).

Accessible: standardised protocols and longevity. The software and its metadata are retrievable using the standardised **HTTPS** protocol (**A1**), which is open, free, and universally implementable (**A1.1**). DPCexplorer is a public resource that requires no login, authentication, or authorisation to explore the data or download biological files (**A1.2**). Because the project metadata is deposited on Zenodo, it remains accessible even if the web application at <https://dpcexplorer.areasciencepark.it> were taken offline (**A2**).

Interoperable: data exchange and linked objects. DPCexplorer exchanges data with other software through domain-relevant community standards, including **FASTA**, **PDB**, **HMM**, **MSA**, and **CSV** formats (**I1**). It also provides qualified references to other digital objects (**I2**): every Pfam ID links to **InterPro/Pfam**, every UniProt accession links to **UniProt**, and every DPCfam AlphaFold representative links to **AlphaFoldDB**.

Reusable: licensing, provenance, and standards. To help researchers assess the suitability of the data for their own work, we provide a range of accurate attributes, including interactive profiling reports for the database tables (**R1**). The project is released under clear licences: the **MIT licence** for the application source code and **CC BY 4.0** for the preprocessed datasets [2] (**R1.1**). We provide detailed provenance by sharing all the preprocessing notebooks and scripts required to transform the raw data into the final results (**R1.2**). The repository lists every dependency in a standard `requirements.txt` file (**R2**), and the software meets

domain-relevant community standards by using the **Django** [30] framework and **PostgreSQL** [32] following modern web development patterns (**R3**).

A programmatic **REST API**, the one remaining gap toward full machine accessibility, is left to future work (Chapter 7).

6.4 Reproducibility

Reproducibility was treated as a first-class concern from the very beginning of the project. From a fresh clone to a running web server, the full stack is rebuilt in five steps, documented in the repository's `README.md` [1] and largely automated by a single setup script.

1. **Clone and create an environment.** Clone the repository and create a Python 3.12 virtual environment. The application code, Django migrations, SQL scripts, and all preprocessing notebooks and scripts live in this single repository.
2. **Install dependencies.** A single `pip install -r requirements.txt` command installs all Python packages the project depends on.
3. **Fetch the data automatically.** The `setup_dpcexplorer_data.sh` script downloads the compressed archives from our data deposit [2] and the original DPCfam repository [12], extracts them into the expected `static/` layout, and validates the file counts on its own. It offers two modes: a **Full installation** (~50 GB, all DPCfam and DPCstruct files) and a **Lightweight mode** (~25 GB, DPCstruct and the CSV tables only) for a fast review.
4. **Initialise the database.** Three pairs of SQL scripts (one pair for each application: `dpc`, `dpcfam`, `dpcstruct`) create all tables, enable the `pg_trgm` trigram extension required for fast Pfam regex search, bulk-load the CSV files with `COPY`, and build the indexes.
5. **Run the server.** A single `python3 manage.py runserver` command starts the application at `http://127.0.0.1:8000/`.

In short, the preprocessed Zenodo repository [2] serves as the reproducibility checkpoint: researchers do not need to download or parse any heavy raw archive to get the application running. Those who prefer to rebuild everything from the raw source files can still do so with the preprocessing notebooks, though that path needs a powerful machine (Table A.1), since the 81 GB DPCfam XML exceeds the memory of a standard workstation. Once the CSV files are loaded into PostgreSQL [32], the database initialisation is fast and works on any computer.

Chapter 7

Conclusion and Future Work

This project started with a concrete problem: two scientifically valuable datasets, DPCfam and DPCstruct, were sitting on Zenodo as raw archives that most researchers had no practical way to explore interactively. Our mission was to change that.

7.1 Summary and Contributions

To achieve our goals, we designed a **five-stage end-to-end pipeline**. The first stage was data acquisition, where we downloaded all source files from Zenodo [12, 13]. The second stage, the most demanding one, was preprocessing. Everything ran on the **ORFEO HPC cluster** through a series of Python notebooks and scripts. For DPCfam, we parsed the XML archives, streamed the 81 GB annotated file to extract protein lengths, Pfam annotations, and domain coordinate ranges, and merged some outputs into consistent CSV dataframes. Similarly, for DPCstruct, we consolidated the metacluster properties, resolved protein lengths through a four-step cascade involving the UniProt REST API, the full Swiss-Prot and TrEMBL FASTA releases, and UniParc, and reorganized the PDB files into per-metacluster subfolders ready for the Mol* viewer. At the end of this stage, we had ten clean, database-ready CSV files, on which we performed a systematic **Exploratory Data Analysis**, covering **81,384** DPCfam (Standard and DPCfamB subsets) and **28,246** DPCstruct metaclusters.

The third stage was database ingestion, where relational metadata were loaded into a **PostgreSQL** database organized into ten tables and optimized with B-Tree, GIN trigram, and functional indexes, later audited through query-plan analysis (Section 6.2), to keep every query fast regardless of table size, while static biological files (FASTA seeds, MSA profiles, HMM models, and PDB structures) were stored separately and served directly from the Django static file system, without any database involvement. The fourth and fifth stages brought everything together: a **Django 6.0.1** backend

exposed the data through four query types (**DPCfam MCID**, **DPCstruct MCID**, **Pfam ID**, **UniProt accession**), with paginated metacluster tables, a graphical domain architecture diagram per protein, and an embedded **PDBe-Molstar** viewer for interactive in-browser visualization of AlphaFold2-predicted structures, colored by pLDDT confidence.

Through this pipeline, we delivered four concrete contributions. First, we built a **unified discovery portal** that removes the technical barrier to entry: anyone can now explore the DPCfam and DPCstruct datasets through a browser without writing a single line of parsing code; the tool will soon be accessible at <https://dpcexplorer.areasciencepark.it/>. Second, we produced a **structured relational resource** that returns complex queries in milliseconds, even across millions of protein assignments. Third, we integrated a **3D visualization tool** that brings AlphaFold2-predicted structures directly into the browser with no external software. Finally, we shared a **fully reproducible workflow**: every script, notebook, schema, and cleaned dataset is publicly available on GitHub and on a dedicated Zenodo deposit [2], inheriting the open licences of the upstream sources [12, 13], so that anyone can reproduce, reuse, or extend our work. Beyond these four, the latest release (v1.0.3) [1] adds a read-only-by-default administration panel for safe data inspection and an automated setup script that turns a fresh clone into a running instance, and we assessed the platform against the **FAIR4RS** principles for research software.

7.2 Future Work and Closing Thoughts

Our work is a small but deliberate step in the ongoing FAIRification of the DPCfam and DPCstruct datasets: *big goal, small steps*. The next person who picks this up has a solid foundation to build on, and three improvements would make the platform significantly more powerful. First, a **REST API** built on the Django REST Framework would make the platform truly machine-accessible, so that a researcher running a pipeline can programmatically fetch JSON metadata or biological files for any MCID, elevating the project from *good FAIR* to *excellent FAIR*. Second, an **in-browser MSA viewer** would let researchers inspect DPCfam Multiple Sequence Alignments (MSAs) without leaving the platform, the same way Mol* already handles PDB files. Third, a **quantitative FAIR assessment** covering the **15 sub-principles** encapsulated within the FAIR framework and producing a measurable, reproducible score would replace the qualitative evaluation of Chapter 6 with something trackable across future releases.

We hope DPCexplorer becomes a practical entry point for the experimental identification of at least some of the putative protein domain families that automatic clustering has brought to the surface.

Bibliography

- [1] E. Nyandu Kagarabi, E. Saadat, and V. Piomponi, “Dpcexplorer: A django web application for interactive exploration of dpcfam and dpestruct protein domain classifications,” June, 2026. <https://doi.org/10.5281/zenodo.20963411>.
- [2] E. Nyandu Kagarabi, E. Saadat, and V. Piomponi, “Preprocessed datasets for interactive exploration of dpcfam and dpestruct protein domain classifications,” May, 2026. <https://doi.org/10.5281/zenodo.20159208>.
- [3] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, “Protein function,” in *Molecular Biology of the Cell. 4th edition*. Garland Science, 2002.
- [4] UniProt Consortium, “Uniprot: the universal protein knowledgebase in 2025,” *Nucleic acids research* **53** no. D1, (2025) D609–D617.
- [5] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with alphafold,” *nature* **596** no. 7873, (2021) 583–589.
- [6] M. Varadi, S. Anyango, M. Deshpande, S. Nair, C. Natassia, G. Yordanova, D. Yuan, O. Stroe, G. Wood, A. Laydon, *et al.*, “Alphafold protein structure database: massively expanding the structural coverage of protein-sequence space with high-accuracy models,” *Nucleic acids research* **50** no. D1, (2022) D439–D444.
- [7] J. Mistry, S. Chuguransky, L. Williams, M. Qureshi, G. A. Salazar, E. L. Sonnhammer, S. C. Tosatto, L. Paladin, S. Raj, L. J. Richardson, *et al.*, “Pfam: The protein families database in 2021,” *Nucleic acids research* **49** no. D1, (2021) D412–D419.
- [8] T. Paysan-Lafosse, A. Andreeva, M. Blum, S. R. Chuguransky, T. Grego, B. L. Pinto, G. A. Salazar, M. L. Bileschi, F. Llinares-López, L. Meng-Papaxanthos, *et al.*, “The pfam protein families database: embracing ai/ml,” *Nucleic acids research* **53** no. D1, (2025) D523–D534.

- [9] E. T. Russo, F. Barone, A. Bateman, S. Cozzini, M. Punta, and A. Laio, “Dpcfam: unsupervised protein family classification by density peak clustering of large sequence datasets,” *PLOS Computational Biology* **18** no. 10, (2022) e1010610.
- [10] F. Barone, A. Laio, M. Punta, S. Cozzini, A. Ansuini, and A. Cazzaniga, “Unsupervised domain classification of alphafold2-predicted protein structures,” *PRX Life* **3** no. 2, (2025) 023009.
- [11] A. Rodriguez and A. Laio, “Clustering by fast search and find of density peaks,” *science* **344** no. 6191, (2014) 1492–1496.
- [12] E. T. Russo and F. Barone, “Metaclusters by dpcfam clustering of uniref50 v 2017_07,” Jan., 2022. <https://doi.org/10.5281/zenodo.6900559>.
- [13] F. Barone, A. LAIO, M. Punta, S. COZZINI, A. Ansuini, and A. Cazzaniga, “Dpcstruct classification of alphafold2-predicted protein structures,” Aug., 2024. <https://doi.org/10.5281/zenodo.13334296>.
- [14] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, *et al.*, “The fair guiding principles for scientific data management and stewardship,” *Scientific data* **3** no. 1, (2016) 1–9.
- [15] European Organization For Nuclear Research and OpenAIRE, “Zenodo,” 2013. <https://www.zenodo.org/>.
- [16] D. Sehnal, S. Bittrich, M. Deshpande, R. Svobodová, K. Berka, V. Bazgier, S. Velankar, S. K. Burley, J. Koča, and A. S. Rose, “Mol* viewer: modern web app for 3d visualization and analysis of large biomolecular structures,” *Nucleic acids research* **49** no. W1, (2021) W431–W437.
- [17] A. Cazzaniga, “Dpcstruct: unsupervised domain classification of alphafold2 predicted structures.” Presented at the *AI in Virology Workshop*, September 9, 2025. Slide deck from Area Science Park, Laboratory of Data Engineering.
- [18] E. T. Russo, *Unsupervised protein family classification by Density Peak clustering*. Phd thesis, Scuola Internazionale Superiore di Studi Avanzati (SISSA), Trieste, Italy, 2020. Physics Area – PhD course in Physics and Chemistry of Biological Systems.
- [19] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, “The protein data bank,” *Nucleic acids research* **28** no. 1, (2000) 235–242.

- [20] I. Barrio-Hernandez, J. Yeo, J. Jänes, M. Mirdita, C. L. Gilchrist, T. Wein, M. Varadi, S. Velankar, P. Beltrao, and M. Steinegger, “Clustering predicted structures at the scale of the known protein universe,” *Nature* **622** no. 7983, (2023) 637–645.
- [21] M. Blum, H.-Y. Chang, S. Chuguransky, T. Grego, S. Kandasaamy, A. Mitchell, G. Nuka, T. Paysan-Lafosse, M. Qureshi, S. Raj, *et al.*, “The interpro protein families and domains database: 20 years on,” *Nucleic acids research* **49** no. D1, (2021) D344–D354.
- [22] I. Sillitoe, N. Bordin, N. Dawson, V. P. Waman, P. Ashford, H. M. Scholes, C. S. Pang, L. Woodridge, C. Rauer, N. Sen, *et al.*, “Cath: increased structural coverage of functional space,” *Nucleic acids research* **49** no. D1, (2021) D266–D273.
- [23] A. Andreeva, E. Kulesha, J. Gough, and A. G. Murzin, “The scop database in 2020: expanded classification of representative family and superfamily domains of known protein structures,” *Nucleic acids research* **48** no. D1, (2020) D376–D382.
- [24] H. Cheng, R. D. Schaeffer, Y. Liao, L. N. Kinch, J. Pei, S. Shi, B.-H. Kim, and N. V. Grishin, “Ecod: an evolutionary classification of protein domains,” *PLoS computational biology* **10** no. 12, (2014) e1003926.
- [25] F. Barone, *Unsupervised methods for protein domain identification and classification*. Phd thesis, Università degli Studi di Trieste, Trieste, Italy, 2024.
- [26] J. MacQueen, “Multivariate observations,” in *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, University of California press Oakland, CA, USA. 1967.
- [27] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *kdd*, vol. 96, pp. 226–231. 1996.
- [28] M. Barker, N. P. Chue Hong, D. S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, *et al.*, “Introducing the fair principles for research software,” *Scientific data* **9** no. 1, (2022) 622.
- [29] Data-Centric AI Community, “fg-data-profiling: Profile your pandas DataFrame,” 2026. <https://github.com/Data-Centric-AI-Community/fg-data-profiling>. A fork of ydata-profiling (formerly pandas-profiling).
- [30] Django Software Foundation, *Django Documentation*, May, 2026. <https://docs.djangoproject.com/en/6.0/>. Version 6.0.6.dev20260507115350.

- [31] S. Chen, S. Ahmmed, K. Lal, and C. Deming, “Django web development framework: Powering the modern web,” *American Journal of Trade and Policy* **7** no. 3, (2020) 99–106.
- [32] The PostgreSQL Global Development Group, *PostgreSQL 18.3 Documentation*, May, 2026. <https://www.postgresql.org/docs/18/>. Version 18.3.

Appendix A

Preprocessing Inventory

A.1 Experimental Environment

Our preprocessing was conducted on the **ORFEO HPC cluster**. Jobs were submitted via the SLURM workload manager to an **epyc** compute node. The technical characteristics of the HPC environment are summarized in Table A.1 below, while the accompanying **Python** notebooks and scripts are listed in Table A.2.

Table A.1: Hardware characteristics and SLURM environment configurations used on the ORFEO HPC cluster for computational tasks.

Characteristic	Details
Physical Node Specifications (epyc)	
Machine Type / Architecture	x86_64 Server Node (Little Endian)
CPU Platform	Dual AMD EPYC 7H12 64-Core Processors (2.6 GHz max)
Node vCPUs	128 (1 thread per core, 64 cores per socket, 2 sockets)
Node RAM	503 GB
Storage	4.8 PB Network Attached Storage (CephFS)
GPUs	None
Allocated Job Environment	
Workload Manager	SLURM
Allocated vCPUs	12
Python Version	3.13.2
Job Launcher	VSCoDe Tunnel

A.2 Inventory

The table below lists every notebook and script produced during the preprocessing stage, as presented in Section 3.3. The main outputs of our preprocessing can be found

in a dedicated Zenodo repository [2]. For a running instance, these outputs are fetched and arranged automatically by the `setup_dpcexplorer_data.sh` script (Chapter 6); the inventory here documents how they were produced in the first place.

Table A.2: Complete inventory of Python notebooks and scripts produced during the preprocessing stage. All files are stored under `static/` in the GitHub Repository [1]:

ID	Filename	Dataset	Output / Purpose
1	<code>parse.sh</code>	DPCfam	Transforms Metacluster XML data into space-separated tables and FASTA files. Note: Must be run separately for Standard and DPCfamB subsets.
2	<code>alphafoldDB.ipynb</code>	DPCfam	Explores <code>alphafoldDB.txt</code> (generated in the previous step (1)), which lists AlphaFoldDB representative structures for some metaclusters in Standard DPCfam. It produces <code>alphafold_dpcfam_reps.csv</code> (see Table 3.1).
3	<code>mclist.ipynb</code>	DPCfam	Conducts basic exploration of the <code>mclist.txt</code> (SSV) file (from (1)) and converts it into <code>mclist.csv</code> . Similarly, every other SSV file listed in the README of [12] is mapped to a notebook of the same name; this applies to both Standard and DPCfamB (e.g., <code>mclistB.ipynb</code>) subsets.
4	<code>final_merged_dataframe.ipynb</code>	DPCfam	Concatenates all sub-dataframes derived from the previous step (3) for Standard DPCFam. For DPCFamB, we used <code>final_merged_dataframeB.ipynb</code> .
5	<code>dpcfam_all_mcs_props.ipynb</code>	DPCfam	Merges the two dataframes derived from the previous step (4) into a master <code>dpcfam_all_mcs_props.csv</code> file (MC Properties - Table 3.2).
6	<code>parse_fasta_files.ipynb</code>	DPCfam	Parses fasta files from (1) and organizes them in a dataframe for the Standard dataset. For DPCFamB, we used <code>parse_fasta_filesB.ipynb</code> .
7	<code>dpcfam_all_mcs_sequences.ipynb</code>	DPCfam	Merges the two dataframes derived from the previous step (6) into a master <code>dpcfam_all_mcs_sequences.csv</code> file (MC Sequences - Table 3.1).
8	<code>extract_uniref_data.py</code>	DPCfam	Streams 81 GB XML; extracts protein IDs, lengths, Pfam and DPCfam domain ranges.

9	<code>exploring_uniref50_data.ipynb</code>	DPCfam	Explores UniRef50 XML outputs from the previous step (8), and produces <code>uniref50_proteins.csv</code> , ... , <code>uniref50_pfam_valid.csv</code> - Table 3.1.
10	<code>dpcstruct_mcs_properties.ipynb</code>	DPCstruct	Produces MC properties master file <code>dpcstruct_mcs_properties.csv</code> (MC Properties - Table 3.3).
11	<code>dpcstruct_mcs_sequences_info.ipynb</code>	DPCstruct	Produces <code>dpcstruct_mcs_sequences.csv</code> - Table 3.1 and per-MC FASTA files.
12	<code>dpcstruct_reps_pdb.ipynb</code>	DPCstruct	Reorganises PDB files into <code>MCID_pdb/</code> hierarchy for Mol* viewer and <code>MCID_pdb.zip</code> for downloads.
13	<code>fetch_uniprot_complete.py</code>	DPCstruct	Queries UniProt REST API - UniProtKB for missing protein lengths.
14	<code>extract_lengths_from_fasta.py</code>	DPCstruct	Extracts lengths from Swiss-Prot and TrEMBL FASTA releases (~55 GB).
15	<code>fetch_missing_lengths.py</code>	DPCstruct	Queries UniProt REST API - UniParc for remaining missing lengths.
16	<code>dpcstruct_all_proteins_lengths.ipynb</code>	DPCstruct	Consolidates protein lengths from all retrieval steps (13 - 16).
17	<code>dpc_protein_lengths.ipynb</code>	Both	Produces proteins master <code>dpc_protein_lengths.csv</code> file - Table 3.1.
18	<code>dpc_pfam_ids.ipynb</code>	Both	Produces Pfam master <code>dpc_pfam_ids.csv</code> file - Table 3.1.
19	<code>dpcstruct_explore_cath_in_depth.ipynb</code>	DPCstruct	Produces cleaned CATH annotation CSV (qc=0.8 - Table 3.1).
20	<code>dpcstruct_explore_scop_in_depth.ipynb</code>	DPCstruct	Produces cleaned SCOP annotation CSV (qc=0.8 - Table 3.1).

Appendix B

The PDB File Format

The **Protein Data Bank format** (.pdb) is a plain-text file that records the 3D coordinates of every atom in a protein structure. Introduced in 1971, it remains the most widely used format for exchanging structural data and is the native format of both the RCSB PDB [19] and the AlphaFold Protein Structure Database [6].

Overall Layout

A PDB file is *column-based* and *fixed-width*: every line is at most 80 characters wide and begins with a six-character **record type** keyword, left-justified and padded with spaces. The major record categories are listed in Table B.1.

Table B.1: Major record categories in the PDB format.

Record	Section	Meaning
HEADER	Title	Molecule class, deposition date, PDB ID
TITLE	Title	Free-text description of the structure
SEQRES	Sequence	Amino acid sequence of each chain
CRYST1	Crystal	Unit cell dimensions and space group
ATOM	Coordinates	Standard amino acid atom coordinates
HETATM	Coordinates	Ligands, water molecules, modified residues
TER	Coordinates	Marks the end of a polymer chain
CONNECT	Connectivity	Explicit bonds (mainly for HETATM atoms)
MODEL / ENDMDL	Coordinates	Encloses one conformer in multi-model files
END	Bookkeeping	Last record in the file

Table B.3: Standard backbone atoms in the PDB format.

Name	Element	Role
N	N	Backbone nitrogen (peptide bond)
CA	C	Alpha-carbon (chiral centre of residue)
C	C	Backbone carbonyl carbon
O	O	Backbone carbonyl oxygen
CB	C	Beta-carbon (first sidechain carbon)

Table B.4: The 20 standard amino acids with their three- and one-letter codes.

Code	1-Ltr	Name	Code	1-Ltr	Name
ALA	A	Alanine	LEU	L	Leucine
ARG	R	Arginine	LYS	K	Lysine
ASN	N	Asparagine	MET	M	Methionine
ASP	D	Aspartate	PHE	F	Phenylalanine
CYS	C	Cysteine	PRO	P	Proline
GLN	Q	Glutamine	SER	S	Serine
GLU	E	Glutamate	THR	T	Threonine
GLY	G	Glycine	TRP	W	Tryptophan
HIS	H	Histidine	TYR	Y	Tyrosine
ILE	I	Isoleucine	VAL	V	Valine

A Mathematical View

Each **ATOM** line provides a point in $\mathbb{R}^3$ (the x , y , z coordinates) labelled with metadata about which atom and residue it represents. The entire PDB file is therefore a **labelled point cloud in three-dimensional space**, which is exactly what a 3D viewer like **PDBe-Molstar** renders on screen (see section 5.5 and subsection 6.1.3).