Next-Generation Protein Search
From BLAST to Embeddings and ANN

Dr. Ioannis Chamodrakas

Dept. of Informatics & Telecommunications
National and Kapodistrian University of Athens

CODEJAM 2026 – 1/4/2026

Next-Generation Protein Search

- **Alignment search:** BLAST and its statistical foundation
- **Representation learning:** Protein Language Models (ESM-2)
- **Scalable retrieval:** Approximate Nearest Neighbor (ANN) search (FAISS)

Central thesis

Remote homology detection is a **representation** + **retrieval** problem: learn an embedding space where functional/structural neighbors become geometrically close, then search that space efficiently.

Audience assumptions (MSc level)

- You know basics of sequence alignment (local/global) and substitution matrices.
- You know basic ML notions (vectors, similarity, train/test, model capacity).
- Goal today: connect **statistical alignment** $\rightarrow$ **learned geometric similarity** $\rightarrow$ **ANN indexing**.

Assumed concepts: quick refresher (why this exists)

- The next slides briefly recap the **baseline concepts** we assume.
- If any of these feel unfamiliar, note them now — we will not re-derive them later.

Scope

Alignment intuition, homology language, vector similarity, and basic complexity.

Protein sequences (minimal biological baseline)

- A protein is a sequence over an alphabet of **20 amino acids**.
- Typical lengths range from tens to thousands of residues.
- Biological roles depend on **folding** and **interaction interfaces**.

Terminology

Residue = amino acid at a position. **Motif** = short conserved pattern. **Domain** = reusable functional/structural unit.

Homology vs similarity (baseline distinction)

- **Homology** = common evolutionary ancestry (binary concept).
- **Similarity** = an observed measurable signal (sequence, structure, embeddings).
- We use similarity to *infer* homology, but they are not the same.

Why this matters today

Remote homologs may have low sequence similarity but remain homologous (and often structurally related).

Local alignment intuition (what BLAST searches for)

- Alignment tries to place two sequences in correspondence by allowing:
  - substitutions (mismatches),
  - insertions/deletions (gaps).
- **Local alignment** finds the best matching *substrings* (not necessarily full-length).

Scoring idea

Score = matches rewarded + mismatches penalized + gaps penalized (often via BLOSUM + gap costs).

Embeddings (baseline ML concept)

- An **embedding** maps an object to a vector: $x \mapsto \mathbf{v} \in \mathbb{R}^d$.
- The goal is geometric: similar objects become nearby vectors.
- For proteins: a sequence becomes a point in a high-dimensional space.

What changes conceptually

Similarity becomes **geometric** (distances/angles), not symbolic (string matching).

Similarity measures we will use

- **Cosine similarity:**

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

- **Inner product:** $\mathbf{u}^\top \mathbf{v}$
- **Key identity:** if vectors are normalized, inner product = cosine.

Practical consequence

FAISS often uses inner product; we normalize vectors to obtain cosine behavior.

Nearest neighbors (what "search" means in vector space)

- Given database vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and a query $\mathbf{q}$:

  retrieve top-k vectors maximizing similarity (or minimizing distance).

- **Exact kNN:** true top-k by full scan.
- **Approximate kNN (ANN):** near-top-k faster.

Core tradeoff

ANN gives speed by allowing small losses in recall.

Big-O intuition (why we care)

- Exact vector search by brute force:

$$\mathcal{O}(Nd) \text{ per query}$$

  where N = database size, d = embedding dimension.
- With millions of proteins, exact scan becomes expensive.

Motivation for ANN

Index structures aim for sublinear candidate selection, then re-ranking.

Evaluation terms we will use (retrieval)

- **Recall@k:**

$$\text{Recall@k} = \frac{|S_{\text{ANN}} \cap S_{\text{exact}}|}{|S_{\text{exact}}|}$$

- **Latency:** time per query (often report p50/p95).
- **QPS:** queries per second (throughput).
- **Memory:** RAM needed for index + vectors/codes.

Why this matters

Index choice is always a recall–latency–memory tradeoff.

Transition: from assumed basics to today's deep dive

- We now have shared vocabulary for:
  - alignment and homology language,
  - embeddings and cosine similarity,
  - kNN/ANN and evaluation metrics.

Next

We start by dissecting **why BLAST fails** in the remote-homology regime, then build the embedding + ANN alternative.

Roadmap

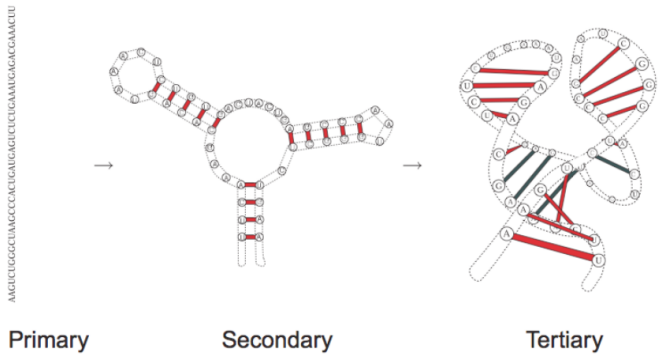
1. 1: Why alignment struggles for remote homologs (BLAST deep dive)
2. 2: ESM-2 and embedding geometry (representation deep dive)
3. 3: ANN and FAISS index design + 15-minute CPU demo

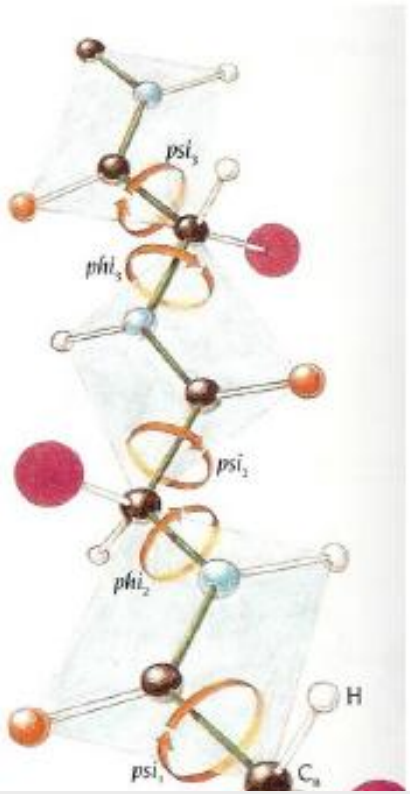
1: Alignment and Remote Homology

Sequence → Structure → Function (why this is hard)

- Sequence similarity is only a **proxy** for biological relatedness.
- Sequence-to-structure relationships are constrained by both physical laws and evolutionary history.
- Structure and function can remain conserved even when sequence diverges.

Why remote homologs matter. Functional similarity may persist even beyond the regime where alignment is reliable.



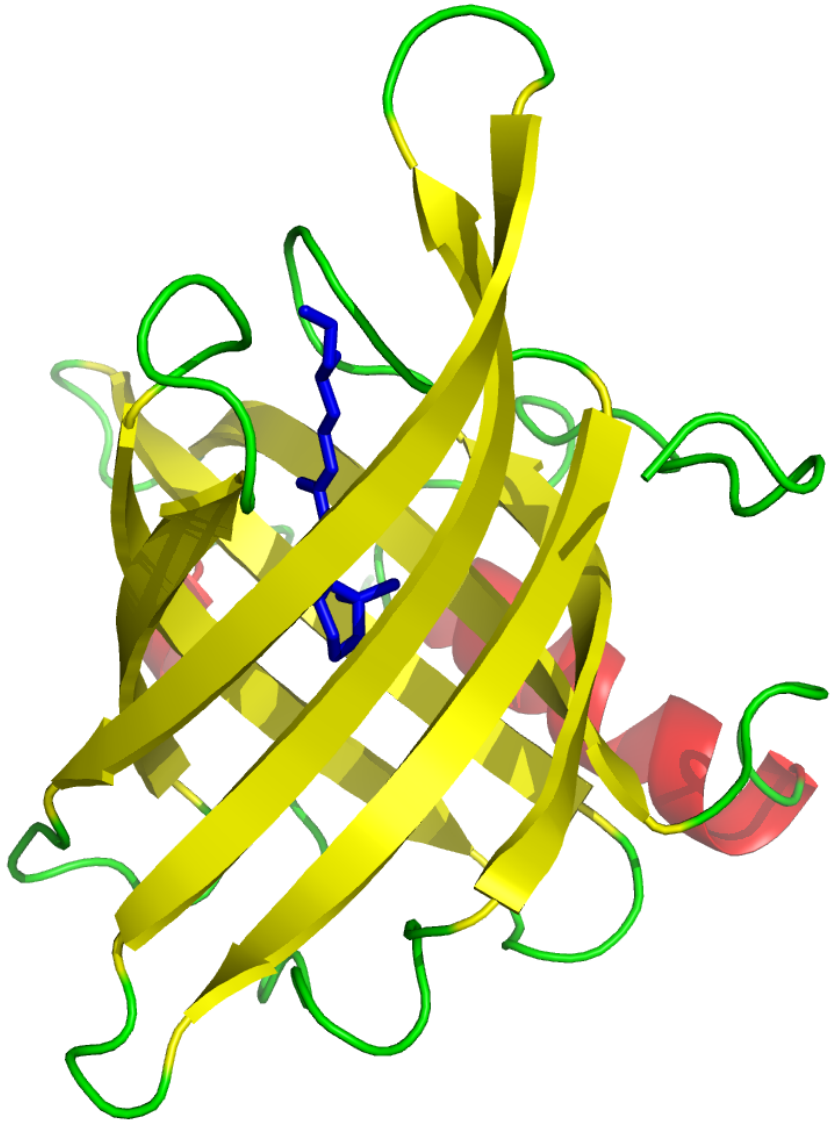


Structure/function can persist under low identity

- Fold-level constraints can dominate over literal letter matching.
- Alignment is strongest when similarity is **local and explicit**.
- Remote homology can be **distributed** across the sequence.

Today's shift

From **string similarity** to **geometric similarity** in embedding space.



Homology: definition (and common confusion)

- **Homology** means common evolutionary ancestry.
- It is not "how similar" two sequences are; it's a **binary relation**.
- We infer homology using evidence: sequence similarity, domains, structure, function.

Remote homologs

Homologous proteins with low sequence identity where alignment evidence becomes weak or ambiguous.

The Twilight Zone (identity regime)

- Rough heuristic (length-dependent):
  - $\geq$ 30% identity: homology often detectable by alignment.
  - 20–30%: Twilight Zone (remote homologs).
  - $<$ 20%: alignment often fails to distinguish from chance similarity.
- This is a **statistical** and **methodological** boundary, not a law of nature.

Why does alignment lose power? (intuition)

- Substitutions accumulate; explicit residue-level correspondence degrades.
- Signal becomes: **weak + distributed + context-dependent**.
- Conserved patterns may be **structural** rather than literal motifs.

Local alignment: what is "locality"?

- Local alignment optimizes over **substrings** (High Scoring Segment Pairs - HSPs).
- It assumes meaningful similarity appears as **contiguous segments**.
- Remote homology may not produce one strong HSP; it can be many weak traces.

BLAST: what problem it solves

- Approximate local alignment at scale.
- Key idea: **seed-and-extend** avoids scanning all alignments.
- It is a brilliant engineering compromise between speed and sensitivity.

BLAST pipeline (conceptual)

1. **Word generation:** query $\rightarrow$ k-mers (and "neighborhood words").
2. **Seeding:** find database hits for words.
3. **Extension:** expand hits into HSPs.
4. **Gapped extension:** allow insertions/deletions.
5. **Statistics:** compute significance (E-value).

Seeds and sensitivity: why k-mers matter

- Seeds are a **filter**: if you miss the seed, you miss the hit.
- Remote homology: fewer exact/near-exact k-mers survive $\Rightarrow$ fewer seeds.
- BLAST tries to compensate via neighborhood words, but there are limits.

Scoring: substitution matrices and gap penalties

- Substitution matrices (e.g., BLOSUM) encode empirical substitution likelihoods.
- Alignment score is additive (matches + mismatches - gaps).
- Additivity is a strong modeling assumption.

Statistics: the role of the E-value

- BLAST approximates the probability that an HSP score occurs by chance.
- **E-value** $\approx$ expected number of random hits with score $\geq S$.
- Lower E-value $\Rightarrow$ more significant.

Interpretation

E-value depends on database size; significance is not an absolute property of a pair of sequences.

Remote homology vs significance

- Remote homologs can yield:
  - no HSP above threshold (false negatives),
  - many weak HSPs that don't "add up" in a way BLAST uses.
- Ranking can also be poor: true remote homologs may appear far below.

Failure mode 1: weak signal spread across the sequence

- Similarity may be weak and spread across many parts of the protein.
- BLAST works best when there is one strong local match.
- In distant homologs, the important signal may come from many small clues together.

Failure mode 2: multi-domain proteins

- Many proteins are made of multiple domains.
- Two proteins may share one domain but differ in other parts.
- Alignment may find the shared domain,
  - but not capture the full functional picture,
  - or be affected by long insertions or repetitive regions.

Failure mode 3: repetitive or low-information regions

- Some regions are repetitive or have little sequence variety.
- Such regions can create misleading similarity signals.
- Filters help, but they do not solve the problem completely.

Key message

Alignment works very well in many cases, but it is less reliable when the sequence signal is weak or distorted.

What comes after BLAST?

- There are stronger methods than basic BLAST:
  - PSI-BLAST
  - profile HMMs (HMMER)
  - domain databases such as Pfam
- These methods use richer models of sequence patterns.
- Today, transformer models go even further and learn general sequence representations.

Part 1: takeaways

- BLAST works very well for close homologs, but remote homology is much harder.
- A main limitation of alignment is that it looks for clear local matches.
- We therefore need representations that capture broader and less obvious sequence patterns.

Transition

We now move from comparing sequences directly to comparing learned embeddings.

2: Protein Language Models and ESM-2

Why Protein Language Models (PLMs) work

- Evolution generates massive unlabeled sequence corpora.
- Functional/structural constraints shape sequence statistics.
- Self-supervised learning can internalize those constraints.

Self-supervision: the core idea

- We learn from sequences **without** explicit labels.
- The training task is a proxy: predict masked residues from context.
- If the proxy is hard, the model must learn rich representations.

Masked Language Modeling (MLM) objective

Given sequence $a_1, \ldots, a_L$ and a masked set M:

$$\max_{\theta} \sum_{i \in M} \log p_\theta(a_i \mid a_{\setminus M})$$

- Model learns conditional distributions constrained by evolution.
- MLM encourages capturing long-range dependencies.

Transformers: the main idea

- Transformers build a representation for each position in the sequence.
- They use **self-attention** so that each position can use information from other positions.
- This helps the model capture patterns that depend on residues far apart in the sequence.

Masked prediction: where does the information come from?

- During training, one residue is hidden and the model must predict it from the rest of the sequence.
- The training signal comes from the original sequence itself: the hidden residue is the target.
- By repeating this over many protein sequences, the model learns which sequence patterns help predict missing residues.
- For a masked position, self-attention lets the model combine information from other positions in the sequence.
- From this context-dependent representation, it predicts a probability distribution over amino acids.

Self-attention: a simple mathematical view

For one position i in the sequence:

$$\text{new representation of position } i = \sum_{j=1}^{L} \alpha_{ij}\, \mathbf{v}_j$$

- We focus on one position i and look at all positions $j = 1, \ldots, L$.
- $\mathbf{v}_j$ is the information contributed by position j.
- α_{ij} is the weight assigned to position j when updating position i.
- Larger α_{ij} means that position j influences position i more strongly.
- The new representation is therefore a weighted combination of information from the whole sequence.

How are the attention weights computed?

$$\alpha_{ij} = \text{softmax}_j\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}}\right)$$

- Consider one position i in the sequence.
- The model compares position i with every position j.
- Each comparison gives a score showing how useful position j is for position i.
- The softmax turns these scores into weights that add up to 1.
- The weight α_{ij} shows how much position j contributes when position i is updated.
- These comparisons are not fixed: they are learned during training.

Self-attention: full picture

$$\text{Attn}(i) = \sum_{j=1}^{L} \alpha_{ij}\, \mathbf{v}_j, \qquad \alpha_{ij} = \text{softmax}_j\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}}\right)$$

- Each position i receives information from all positions j.
- The attention weights α_{ij} determine how strongly each position contributes.
- The result is a context-dependent representation of position i.

Multi-head attention: multiple “views”

- Multiple heads learn different interaction patterns.
- Heads can specialize:
  - local motifs,
  - secondary structure tendencies,
  - long-range coupling.

From training to embeddings

- During training, the model learns parameters that help predict masked residues.
- After training, we can feed a protein sequence through the model.
- Each layer updates the representation of every residue using sequence context.
- The final layer outputs one vector for each residue.
- These output vectors are the residue embeddings.

ESM-2: what it outputs

Given a sequence of length L:

- The model outputs one embedding for each residue: $\mathbf{h}_1, \ldots, \mathbf{h}_L \in \mathbb{R}^d$
- These are contextual residue embeddings.
- For retrieval, we often need a single vector for the whole protein: $\mathbf{v} \in \mathbb{R}^d$

Contextual embeddings (conceptual point)

- A residue is not "Cys = always Cys".
- Its meaning depends on micro-environment and long-range constraints.
- PLMs encode that via contextual hidden states.

Pooling: from residue vectors to protein vector

Mean pooling:

$$\mathbf{v} = \frac{1}{L} \sum_{i=1}^{L} \mathbf{h}_i$$

- Simple, stable, often surprisingly strong.
- But it can wash out multi-domain structure.

Similarity metrics in embedding space

- Cosine similarity:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

- Euclidean distance (often after normalization)
- Inner product (commonly used with FAISS; cosine via normalization)

Geometry: embedding spaces as manifolds

- High-dimensional vectors often lie on lower-dimensional manifolds.
- Nearby vectors can correspond to:
  - similar folds,
  - similar biochemical roles,
  - shared domain content.

What does “neighbor” mean biologically?

- Neighbor = small distance under chosen metric.
- But biological relatedness requires validation:
  - domains (Pfam),
  - Gene Ontology (GO) terms,
  - Enzyme Commission (EC) numbers,
  - UniProt descriptions,
  - structure (when available).

Remote homology through embeddings: the logic

1. Query protein $\rightarrow$ embedding $\mathbf{q}$
2. Retrieve nearest neighbors in embedding space
3. Check whether BLAST misses some of them or ranks them poorly
4. Validate with orthogonal evidence (domains/GO/structure)

Validating a retrieved neighbor: a worked example

Step 1 — Is it reviewed?

- Check whether the hit is in **SwissProt** (manually reviewed) or only **TrEMBL** (computationally predicted).
- TrEMBL annotations are weaker evidence.

Step 2 — Compare domains

- Look up both query and hit in **Pfam** (or InterPro).
- Shared domain architecture ⇒ stronger support.
- Shared one domain but different overall architecture ⇒ partial homology only.

Step 3 — Compare function

- **EC number:** same 4-digit code ⇒ same reaction; same first digit only ⇒ weak support.
- **GO terms:** compare Molecular Function and Biological Process; identical terms are strong signal, ancestor-only overlap is weaker.

Step 4 — Synthesize

- Agreement across domain + EC + GO ⇒ plausible remote homolog.
- Single-source agreement only ⇒ treat as a hypothesis, not a finding.

Key reminder

Annotations in TrEMBL are often transferred automatically. Validating a retrieved neighbor with predicted annotations does not provide independent evidence.

Failure modes of PLM embeddings

- **Convergent evolution:** similar function without ancestry.
- **Composition bias:** neighbors due to amino-acid distribution.
- **Multi-domain averaging:** mean pooling blends unrelated domains.
- **Length effects:** padding/truncation, uneven contributions.

Mitigations / best practices

- Normalize vectors (cosine) to reduce scale effects.
- Consider domain-level embeddings when feasible.
- Filter low-complexity sequences.
- Validate with annotation and, when possible, structure.

Part 2: takeaways

- ESM-2 turns sequences into contextual vectors.
- Similarity becomes geometric; retrieval becomes nearest-neighbor search.
- The remaining challenge: search **fast** at large scale.

Transition

Now we treat retrieval as an ANN indexing problem: recall vs latency vs memory.

3: ANN, FAISS, and Index Design

The retrieval problem at scale

- Database: N vectors in $\mathbb{R}^d$ (often $d \in [320, 1280]$)
- Query: given $\mathbf{q}$, find top-k nearest vectors
- Exact search cost: $\mathcal{O}(Nd)$ per query (too slow for large N)

Why high dimensions are hard

- **Distance concentration:** nearest and farthest become similar.
- **Index degeneracy:** trees (kd-tree) lose pruning power.
- Approximation becomes necessary.

ANN: controlled approximation

- We accept approximate neighbors for huge speedups.
- Performance is a tradeoff between:
  - Recall@k,
  - query latency (ms),
  - throughput (QPS),
  - memory footprint.

Evaluation metrics for ANN retrieval

- Recall@k:

$$\text{Recall@k} = \frac{|S_{\text{ANN}} \cap S_{\text{exact}}|}{|S_{\text{exact}}|}$$

- QPS (queries per second)
- Latency percentiles (p50, p95)

ANN families (map)

- **Hashing-based:** LSH (probabilistic guarantees)
- **Partition/quantization-based:** IVF, PQ
- **Graph-based:** HNSW

LSH: the core principle

- Use a family of hash functions $h(\cdot)$ such that:
  - nearby points collide with high probability,
  - far points collide with low probability.
- Use multiple hash tables to increase recall.

LSH tradeoffs

- Pros: theory-friendly, sublinear expected queries.
- Cons: memory-heavy; many tables needed for high recall; tuning is nontrivial.
- In practice, many systems prefer IVF/HNSW for embeddings.

IVF: inverted file index (coarse quantization)

- Train n_{list} centroids via k-means (coarse quantizer).
- Assign each DB vector to nearest centroid (inverted list).
- Query: find nearest centroids, scan only their lists.

PQ inside IVF: compressing the vectors in each list

- The inverted lists can still contain many vectors.
- Instead of storing full float vectors, we compress them with **product quantization (PQ)**.
- Each vector is split into m subvectors and each subvector is quantized separately.
- So each list stores compact codes, making search faster and more memory-efficient.

HNSW: graph-based approximate search

- Represent the database as a graph, where each vector is connected to nearby vectors.
- Query: start from an entry point and move step by step toward closer vectors.
- This avoids scanning large parts of the database.

Which ANN family when? (practical guidance)

- **Flat**: exact (fast enough for small N), great baseline.
- **IVF**: strong for large N.
- **IVF+PQ**: when memory is the bottleneck.
- **HNSW**: excellent recall/latency; memory heavier than PQ.
- **LSH**: hashing-based alternative, but less often a first choice for dense embeddings.

What do we do with FAISS in practice?

- Store protein embeddings in a searchable index
- Choose an index type: exact or approximate
- Add database vectors once, then issue many nearest-neighbor queries
- Retrieve the top-k most similar proteins for each query

Exact search baseline in FAISS

- If vectors are normalized, inner product is equivalent to cosine similarity.
- `IndexFlatIP` compares the query against all database vectors.
- This gives the exact nearest neighbors.
- ANN indexes are then compared against this exact baseline.

How do we choose a FAISS index?

- **Flat**: use when the database is small enough for exact search
- **IVF**: use when the database is large and exact search is too slow
- **IVF+PQ**: use when memory is also a major constraint
- **HNSW**: use when very strong recall/latency tradeoffs are needed

Key point

Index choice depends on scale, latency targets, memory budget, and required recall.

How do we evaluate a FAISS-based retrieval system?

1. Fix the embedding representation
2. Compute exact neighbors as a reference
3. Build and tune the FAISS index
4. Measure recall, latency, throughput, and memory

Two different evaluation questions

- **System question:** does the ANN index recover the exact embedding neighbors?
- **Biological question:** are the retrieved neighbors meaningful in terms of homology or function?
- These are related, but they are not the same question.

Where does BLAST enter the evaluation?

- BLAST is not the ground truth of the embedding space.
- It is a strong biological baseline for comparison.
- We ask whether embedding neighbors:
  - overlap with BLAST hits,
  - extend beyond BLAST in plausible ways,
  - or produce misleading matches.

How should we interpret retrieval results?

1. **Agreement with BLAST:** both methods recover similar neighbors.
2. **Useful extension beyond BLAST:** embeddings retrieve plausible additional neighbors.
3. **Possible failure cases:** embeddings retrieve biologically weak or misleading neighbors.

Live Demo (15 minutes, CPU-only)

Demo roadmap

1. Load a small pretrained ESM-2 model on CPU
2. Compute embeddings for a tiny sequence collection
3. Build an exact FAISS cosine index
4. Run a few queries and inspect the retrieved neighbors
5. Briefly connect this exact search to ANN scaling ideas

What this demo is meant to show

- This is a **pipeline demo**, not a biological benchmark.
- We use a pretrained model: no training happens here.
- Focus on the main stages:
  - sequence $\rightarrow$ embedding
  - embedding $\rightarrow$ index
  - index $\rightarrow$ nearest neighbors

Demo: installation (CPU)

```
pip install "torch>=2.0" transformers accelerate
pip install faiss-cpu
```

Demo code (1/3): load model and toy data

```
import torch
import numpy as np
import faiss

# HF ESM-2 classes
from transformers import (AutoTokenizer,
     EsmModel)

# small CPU-friendly model
MODEL = "facebook/esm2_t6_8M_UR50D"
device = "cpu"
tok = AutoTokenizer.from_pretrained(MODEL)

# Load base model without pooling
mdl = EsmModel.from_pretrained(
    MODEL,
    add_pooling_layer=False
).to(device)
mdl.eval() # inference mode

TOY_DB = [
    # Family A: P-loop-like
    ("A001", "p_loop_A",
     "MNNIRRVLIVGPNGAGKSTLLQAIAA..."),
    ("A002", "p_loop_A",
     "MSNIRRVLIVGPNGAGKSTLLQAVAA..."),
    # Family B: helix-turn-helix-like
    ("B001", "hth_B",
     "MARRKQLAERLAALEQQNPDVEALAALE..."),
    ("B002", "hth_B",
     "MARRKQLAERLAALEQQNPDVEALASLE..."),
    # Family C: acidic enzyme-like
    ("C001", "acidic_C",
     "MNKDVAIHFDLSPEDVKRALEAGADVVV..."),
    ("C002", "acidic_C",
     "MNKDVAIHFDLSPEDVKRALEAGADIVV..."),
    # Family D: ubiquitin-like
    ("D001", "ubq_like_D",
     "MQIFVKTLTGKTITLEVEPSDTIENVKAKI..."),
    ("D002", "ubq_like_D",
     "MQIFVKTLTGKTITLEVEPSDTIENVKAKI..."),
    # Hydrophobic like
    ("X001", "hydrophobic_like",
     "MKRLLPLAVAVAALLAVSCSAQAAAAPAAEAEAAAG"),
    # Low-complexity / compositionally biased
    ("X003", "low_complexity",
     "MSDSEEKKTKKTKKTKKTKKTKKTKKTKKTKK")
]
```

Demo code (2/3): embeddings

```
@torch.no_grad()
def embed_batch(seqs):
    # tokenize batch
    inp = tok(
        seqs,
        return_tensors="pt",
        padding=True,
        truncation=True
    )
    # move to CPU
    inp = {k: v.to(device) for k, v in inp.items
     ()}
    out = mdl(**inp)
    h = out.last_hidden_state

    # mask padded tokens
    attn = inp["attention_mask"].unsqueeze(-1).
     float()

    # mean pool
    v = (h * attn).sum(dim=1) / attn.sum(dim=1).
     clamp(min=1.0)
    # L2 normalize
    v = torch.nn.functional.normalize(v, dim=1)
    # FAISS expects float32
    return v.cpu().numpy().astype("float32")
```

```
# Unpack the database
ids = [
    sid for (sid, lbl, seq) in TOY_DB
]
labels = [
    lbl for (sid, lbl, seq) in TOY_DB
]
seqs = [
    seq for (sid, lbl, seq) in TOY_DB
]

# Compute database embeddings
X = embed_batch(seqs)
d = X.shape[1] # dimension
```

Demo code (3/3): exact FAISS retrieval

```
# exact inner-product index
index = faiss.IndexFlatIP(d)
index.add(X)

def search(query_seq, k=5):
    # embed query
    q = embed_batch([query_seq])
    # search top-k
    scores, nn = index.search(q, k)

    out = []
    # pair ids and scores
    for j, s in zip(nn[0], scores[0]):
        out.append((
            ids[j],
            labels[j],
            float(s),
            len(seqs[j])
        ))
    return out
```

```
query = seqs[0] # example query
hits = search(query, k=5)

print("Query length:", len(query))

# pretty print results
for r, (sid, lbl, score, L) in enumerate(hits,
      1):
    print(
        f"{r:>2}. {sid}  "
        f"label={lbl:>15}  "
        f"cosine={score:.3f}  "
        f"len={L}"
    )
```

What to look for in the output

- Does the top hit correspond to the query itself?
- Do nearby sequences come from the same toy group?
- Do low-complexity or hydrophobic sequences create suspicious neighborhoods?
- How quickly do cosine scores fall after the first few hits?

Suggested live queries

- Query with a sequence from **family_A**: inspect whether the nearest non-self hit stays nearby
- Query with a **hydrophobic_like** sequence: discuss composition-driven neighborhoods
- Query with **low_complexity**: discuss spurious similarity and the need for validation
- Emphasize: cosine similarity is a **retrieval signal**, not biological proof

From Demo to Scale: The Need for ANN

- `IndexFlatIP` is exact, but it scans the whole database.
- This is fine for tiny collections (like our toy demo), but fails for millions of proteins.
- At scale, we replace exact search with ANN indexes such as IVF or HNSW.
- The main engineering question becomes: how much speed do we gain, and how much recall do we lose?

Wrap-up

Synthesis: three layers of the story

- **Alignment layer:** explicit local correspondence and statistics
- **Representation layer:** contextual embeddings and learned geometry
- **Retrieval layer:** scalable search in embedding space

What to remember

- Remote homology is a setting where inductive biases matter strongly.
- Embeddings define a new geometric space for similarity.
- FAISS and ANN methods make that space searchable at scale.

One-line summary

We move from comparing strings directly to searching a learned geometric space of proteins.

Possible extensions

- Compare BLAST, PSI-BLAST, HMMER, and ESM-based retrieval on the same benchmark
- Compare Flat, IVF, and HNSW on the same embedding set
- Study the effect of pooling strategy and layer choice