ChatGPT

Retrieval-Augmented Generation with ESM for PETase Engineering

Using ESM Embeddings to Propose Novel But Similar PETase Variants

ESM Embeddings as Sequence Signatures: Large protein language models like ESM-1b/ESM-2 transform sequences into high-dimensional embeddings that capture subtle functional and structural relationships [1]. Unlike raw sequence identity, which may miss distant yet functionally similar homologs, ESM embeddings encode evolutionary constraints and protein family features. In fact, similarity in embedding space correlates only weakly with sequence identity, enabling recognition of remote homologs (even <20% identity) via embedding-based comparison [1]. This means PETase variants known to be active or stable will cluster in ESM's embedding space; new sequences that fall in or near this cluster are likely to preserve the PETase fold and function.

Retrieval-Augmented Proposal of Variants: Instead of random mutagenesis, you can leverage a database of known PETase or cutinase variants (and related enzymes) embedded with ESM. Using a similarity search (e.g. k-nearest neighbors in embedding space), retrieve sequences that are closest to the wild-type or current lead variant. These retrieved sequences might include FAST-PETase or other engineered mutants with improved stability. By examining their differences from wild-type, you can identify candidate mutations or motifs that have previously conferred stability. Retrieval-augmented generation (RAG) would then use these "neighbor" sequences as guides to propose new variants that are **novel yet similar** to known good variants. For example, if multiple top-neighbors share a mutation (e.g. a certain residue substitution that rigidifies a loop), the model can propose adding that mutation to the wild-type. This approach injects **knowledge-driven mutations**: the proposals aren't random, but drawn from patterns the language model recognizes as plausible and beneficial. Essentially, the ESM embedding acts as a **context filter**, constraining variants to those **close in latent space** to proven functional sequences. Recent work on RAG-ESM (a retrieval-conditioned ESM2) demonstrates this principle: conditioning on a homologous sequence guides a transformer to generate proteins in the same family, with outputs converging to embedding-space regions near the context sequence [2]. In practice, you could implement a simplified version by feeding the wild-type and a retrieved variant into an ensemble procedure – for instance, aligning them and swapping some segments or using the retrieved sequence as a template for masked language modeling. This ensures proposed variants stay in-family and likely **preserve key structural motifs**, while still introducing novel combinations of mutations inspired by the retrieval set.

Neighborhood Search in Embedding Space for Local Mutations

Defining an Embedding-Space Radius: ESM embeddings provide a quantitative way to define a "neighborhood" around the wild-type sequence. You can use a distance metric (e.g. cosine or Euclidean distance on the embedding vectors) to measure how far a mutant is from the wild-type in latent space. To favor minimal perturbation, constrain candidate mutations to those that keep the embedding within a

certain radius of the wild-type. For example, single-residue substitutions that are conservative (chemically similar) will typically yield small shifts in the ESM embedding, whereas a large insert or many mutations would shift the embedding more dramatically. By **selecting mutations with small embedding distances**, you ensure the variant remains close to the wild-type's functional manifold. This approach aligns with a *trust-region* idea: explore locally around the wild-type in embedding space, analogous to staying within a few steps on the evolutionary landscape. Notably, ESM embeddings have been shown to capture functional/structural similarity beyond what sequence identity alone captures [1], so a small movement in this space likely indicates the mutant still folds and functions similarly.

Balancing Similarity and Diversity: Within this embedding-defined neighborhood, you can introduce **beneficial diversity** by sampling different directions. One strategy is to generate all single mutants (or single + double mutants) of PETase, compute their embeddings via ESM, and filter to those within a tight distance threshold of the wild-type. Among these, you might further select ones that your stability model predicts to improve thermostability (e.g. $\Delta\Delta G < 0$). These are "local" moves – close in sequence and embedding – so they likely maintain activity. By combining a few such mutations, you can then propose multi-mutation variants that still reside near wild-type in embedding space. Importantly, the ESM embedding distance can serve as a **regularizer**: if a candidate's embedding drifts too far, it may indicate a change that risks misfolding or altering specificity, so those candidates could be pruned or deprioritized. This helps maintain a focus on *quality* (retaining the PETase core functionality) while exploring *diversity* in the immediate vicinity of the wild-type. Over iterative rounds, you can gradually expand the radius – initially require candidates to be very close to wild-type's embedding, and later allow slightly larger shifts as you accumulate confidence in the surrogate model's predictions. This ensures a **controlled exploration**: small, safe jumps first, then progressively wider exploration once you've found footholds of stability.

Combining QD + BO with Embedding-Based RAG

Quality-Diversity (QD) Overview: The QD + BO strategy from the PDF maintains an archive of diverse high-performing variants across different "behavioral niches." In PETase engineering, niches might be defined by properties like mutation count or predicted stability range [3]. The goal is to avoid converging on a single optimum by instead preserving multiple candidates that represent trade-offs (e.g. some with ultra-high stability but moderate activity, others with slightly lower stability but near-native activity). Bayesian optimization (BO) then guides selection of which candidates to actually test, using an uncertainty-aware acquisition function to balance exploitation of the surrogate model vs. exploration of uncertain regions.

RAG-Driven Candidate Generation: Embedding-space RAG can be naturally integrated into the *candidate generation* phase of each design-build-test-learn cycle. The attached framework already suggests using a protein language model or MPNN to generate plausible mutants rather than random ones [3]. Here, ESM with RAG can fulfill that role. Specifically, for each niche in the QD archive, you could **retrieve** a set of sequences relevant to that niche's current best (for example, find homologous sequences or known variants close to the archive elite in embedding space) and use those as context to **generate new variants**. This could mean taking an elite sequence and using ESM to produce single-site variants that ESM deems plausible (high language model probability) or using a RAG approach to condition on a homolog of that elite. By doing so, you maintain the **diversity** (each niche gets its own tailored proposals) while also **steering proposals toward promising regions** informed by known functional proteins. For instance, if one niche corresponds to "high stability, 3 mutations" and the elite in that niche is a particular variant, you might retrieve other triple mutants or homologs with high stability embeddings, then have ESM suggest similar triple mutants that haven't been tried. RAG-ESM results support this strategy: conditioning on a

context sequence with desired properties biases the generation toward that part of sequence space [2], effectively pulling the model toward sequences that *resemble known functional ones* (here, those with high stability). This means your generated variants are more likely to **inherit the functional traits** of the context (like thermostability or maintained activity), as opposed to unguided random mutations which could drift into non-functional space.

Steering with Embedding Metrics: In the QD+BO loop, you can also incorporate embedding-based metrics explicitly. For example, to ensure the batch of sequences selected for testing in a round are diverse, one of the diversity measures can be the pairwise distance in ESM embedding space (not just Hamming distance or mutation count). The PDF mentions "explicit sequence-distance diversity controls" – an embedding distance could serve as a sophisticated measure of sequence distance, capturing functional divergence. Conversely, if a certain niche is under-explored, you might allow proposals that are a bit farther in embedding space (to inject novelty) but still retrieve from sequences in the general enzyme family to avoid loss of function. This corresponds to the "novelty injections subject to constraints" idea [3] – the constraints can include staying within the same embedding cluster or preserving catalytic residues. Meanwhile, the Bayesian surrogate model itself can be enhanced by embedding features: indeed, the pipeline plans to feed protein-LM features into the model [4]. That means the ESM embedding (or per-residue ESM features) for a sequence could be part of the input to the stability/activity predictor, improving its accuracy in ranking candidates. In summary, **embedding-based RAG** and QD+BO are complementary: RAG generates candidates that are bias-corrected toward likely-functional regions, and QD+BO evaluates and ensures we explore multiple such regions. Together, they steer the search toward high-value sequences that are both **innovative and credible** (i.e., not too far from what evolution or known experiments tell us works).

Tools and Workflow for Embedding-Based RAG Implementation

Implementing this approach requires combining protein ML libraries with similarity search tools:

- **ESM from FAIR (Facebook AI Research):** The `fair-esm` package (or the equivalent HuggingFace transformers model for ESM-2) provides pre-trained models to obtain sequence embeddings and probabilities. For example, you can load a model like ESM-1b or ESM-2 (650M or larger) and feed in a protein sequence to get the final hidden state for each residue. A common practice is to take the average of all residue embeddings (or use the special [CLS] token if available) to derive a fixed-length vector representation of the entire protein [5]. This vector (typically 1280 dimensions for ESM-1b/ESM-2 models [6]) serves as the sequence's embedding in latent space. You can obtain it in Python via the ESM API or HuggingFace pipeline. For example:

```
import esm
model, alphabet = esm.pretrained.esm2_t33_650M_UR50D()
batch_converter = alphabet.get_batch_converter()
batch = [("petase_wt", wild_type_sequence)]
tokens = batch_converter(batch)[2]  # get token IDs
with torch.no_grad():
    out = model(tokens, repr_layers=[33])
repr = out["representations"][33]  # per-residue embeddings from final layer
seq_embedding = repr.mean(dim=1)   # mean pooling to get a single vector
```

This vector `seq_embedding` is the ESM embedding for wild-type PETase. You would repeat this for all sequences in your database of known variants or homologs to build an embedding library.

- **FAISS for Similarity Search:** With a library of embedding vectors, Facebook's **Faiss** is a powerful tool to perform fast nearest-neighbor search in high-dimensional spaces. It's well-suited for searching millions of vectors, but also works for smaller sets and provides flexibility in distance metrics (dot product, L2, etc.). You can use Faiss to index your ESM embeddings of known sequences and then query it with the wild-type's embedding (or any query sequence's embedding) to retrieve the top-k most similar sequences. Under the hood, Faiss can use algorithms like hierarchical navigable small-world graphs or product quantization for efficiency [7], but from a user perspective, it might look like:

```
import faiss
import numpy as np
# Assume embeddings_matrix is an NxD numpy array of N sequence embeddings
(D=1280)
index = faiss.IndexFlatL2(embedding_dim)  # L2 distance index (can also do
cosine sim)
index.add(embeddings_matrix)
D, I = index.search(np.array([query_embedding]), k=10)  # find 10 nearest
neighbors
nearest_indices = I[0]
nearest_sequences = [sequence_db[i] for i in nearest_indices]
```

This yields the sequences most similar to your query in embedding space. Those could be fed into the RAG pipeline – for instance, take their mutations as suggestions for new variants, or supply one of them as context to a conditional generation model. Faiss ensures this retrieval step is fast even if your database is large (e.g. searching within all PETase-like sequences from UniProt).

- **Sentence-Transformers / Embedding Utilities:** The **sentence-transformers** library is commonly used for text embeddings, but it's also useful for managing embedding workflows in bioinformatics. You could wrap a protein embedding model in a `SentenceTransformer` interface to leverage convenient functions for batching, similarity computation, or even FAISS integration. For example, you might use `SentenceTransformer.encode()` on a list of protein sequences (after installing a protein model) to get their embeddings in one go. Additionally, sentence-transformers provides utilities like `util.semantic_search` which under the hood can perform a cosine similarity search between a query embedding and a corpus of embeddings. While not specialized for proteins, it demonstrates the same concept: **embed** sequences and **search** for neighbors. In practice, many researchers use HuggingFace Transformers directly for protein models – e.g., `AutoModel.from_pretrained("facebook/esm2_t33_650M_UR50D")` – and then use libraries like **NumPy** or **scikit-learn** (for small data) or Faiss (for large data) for similarity queries. The exact choice of tool depends on scale: for a few thousand known variants, a brute-force cosine similarity via NumPy is fine; for millions of sequences, Faiss or an ANN (approximate nearest neighbor) library is essential [7].

- **Integration and Workflow:** Bringing it together, a possible **workflow** is: *(1)* Embed the wild-type PETase and all candidate or reference sequences using ESM. *(2)* Use Faiss to retrieve nearest neighbors of the wild-type (or of the current design candidates) to get sequences that are known to be functional and close in sequence space. *(3)* Feed these retrieved sequences into a generation step – this could be as simple as recommending their unique mutations as variants to test, or as complex as using a model like RAG-ESM to conditionally generate a new sequence. For instance, you could take a top neighbor and create a hybrid: apply one or two mutations from that neighbor onto the wild-type (if they're not already present) to form a novel variant that "stays in the family." Alternatively, use a masked language modeling approach: concatenate the wild-type sequence, mask one or more positions, and let ESM (optionally conditioned on a neighbor sequence in a prompt-like manner) fill in a likely amino acid. This is akin to how one might use GPT with retrieved documents – here the neighbor sequence is the document, and PETase sequence is the context to be "edited" with plausible mutations.

- **Verification and Iteration:** Once variants are proposed, they loop back into the QD+BO pipeline: evaluate their predicted stability/activity via the surrogate, update the quality-diversity archive, and repeat. Each cycle can retrieve fresh neighbors (or even use neighbors of neighbors) to keep expanding the search space intelligently. Throughout, the **ESM embedding serves as a guide** – any time a proposed sequence veers too far (e.g., an out-of-cluster embedding), you might flag it as risky. Conversely, if the surrogate suggests a certain out-of-cluster sequence could be extremely good (high predicted stability), you might then deliberately retrieve sequences near that one's embedding to see if that region of space harbors other viable designs.

In summary, **retrieval-augmented generation with ESM** adds a knowledge-driven layer to your PETase engineering pipeline. It generates candidates that are grounded in known protein sequence-function relationships (through retrieval) and kept biologically plausible by the language model's understanding. By coupling this with the QD+BO framework, you ensure that you not only search **wide** (exploring many diverse niches with QD) but also search **smart** (using ESM embeddings to focus on sequences likely to fold and function). This synergistic approach should yield novel PETase variants that are both **innovative** and **credible**, accelerating the discovery of enzymes with improved thermostability and robust activity [3] [2]. The use of modern libraries like ESM (for embeddings/LM generation), Faiss (for fast neighbor lookup), and others provides a practical path to implement this RAG-enhanced design loop in Python. Each component – from embedding extraction [5] to nearest-neighbor search [7] to conditional sequence generation – is supported by available packages, making it feasible to integrate into your pipeline. By replacing unguided random mutations with **embedding-guided proposals**, you leverage the power of big data (learned by ESM from millions of proteins) to inform each mutation choice, greatly improving the odds of finding beneficial PETase variants in a vast sequence space.

Sources: Supporting information and examples were drawn from recent literature on protein language models and design strategies [3] [1] [2] [7], including the RAG-ESM framework which demonstrates how conditioning on retrieved sequences keeps generated proteins close to a desired family in embedding space [2]. These approaches underscore the value of ESM embeddings for capturing protein similarity and guiding explorations of sequence space in enzyme engineering.

---

1 5 6 7 Frontiers | Nearest neighbor search on embeddings rapidly identifies distant protein relations
https://www.frontiersin.org/journals/bioinformatics/articles/10.3389/fbinf.2022.1033775/full

2 (PDF) RAG-ESM: Improving Pretrained Protein Language Models via Sequence Retrieval
https://www.researchgate.net/publication/394757904_RAG-ESM_Improving_Pretrained_Protein_Language_Models_via_Sequence_Retrieval

3 4 Petase Design Draft.pdf
file://file_00000000a48071f5b95b92bf741c5984