In [1]:

```
import numpy as np
import pandas as pd
import pickle
from Bio import SeqIO
import ipdb
import random
random.seed(10)
import re



class DataFormat():
    '''Class for merging all of the data from the phagescope database

    ***Put all files into one folder with no changes in the file names***

    ***file Names:

    -TemPhD # folder with all of the fasta files
    -TemPhD_antimicrobial_resistance_gene_data.tsv
    -temphd_phage_annotated_protein_meta_data.tsv
    -temphd_phage_meta_data.tsv
    -TemPhD.gff3

    ******'''

    def __init__(self,folderpath, DBName):
        '''Intitialization takes the name of the database data and the pth to the folder

        self.path = folderpath #Path to data folder
        self.DB = DBName #Name of database


        self.amr_tsv = '%s_antimicrobial_resistance_gene_data' % self.DB
        self.protein_tsv = '%s_phage_annotated_protein_meta_data' % self.DB.lower()
        self.gff3_file = '%s' % self.DB
        self.meta_tsv = '%s_phage_meta_data'% self.DB.lower()


        self.data_df = None #Data frame with all of the merged data
        self.fasta_path = '%s/%s' % (self.path,self.DB)#fasta folder path

    #Functions for loading and writing files
    def __load_tsv__(self,file_name):
        '''Function for loading .tsv files'''

        path = '%s/%s.tsv' % (self.path,file_name)
        df = pd.read_csv(path,sep="\t",dtype=str)
        return df

    def __load_gff3__(self,file_name):
        '''Function for loading .gff3 file'''

        path = '%s/%s.gff3' % (self.path,file_name)
        df = pd.read_csv(path,sep="\t",header=None,comment="#",names=["seqid", "source",
                         "score", "strand", "phase", "attributes"])
        return df
```

```
    def __load_fasta__(self,file_name):
        '''Function for loading .fasta file'''

        path = '%s/%s.fasta' % (self.fasta_path,file_name)
        fast_list = list(SeqIO.parse(path, "fasta"))

        df = pd.DataFrame([{"id": rec.id,"description": rec.description,"sequence": str(
                            "length": len(rec.seq)} for rec in fast_list])
        return df

    def load_data_pickle(self, file_name):
        '''Load Pickle File'''

        path = '%s/%s.pickle' % (self.path,file_name)
        f = open(path, 'rb')
        data = pickle.load(f)
        f.close()

        return data

    def savePickle(self,file_name,data):
        '''Save Data as Pickle'''

        path = '%s/%s.pickle' % (self.path,file_name)
        f = open(path, 'wb')
        pickle.dump(data, f)
        f.close()

        return


    def export_csv(self,file_name,data):
        '''export data as .csv'''

        path = '%s/%s.csv' % (self.path,file_name)
        data.to_csv(path)

        return

    #Tools

    def reverse_complement(self,seq):
        """Return the reverse complement of a DNA sequence."""
        complement = str.maketrans("ACGTacgt", "TGCAtgca")
        return seq.translate(complement)[::-1]

    def __get_Phage_seqs__(self,phage_ids,protein_df):
        '''returns all phage sequences and protein sequences from list of phage_ids'''


        df_seqs = self.__load_fasta__(phage_ids[0])
        for x in range(1, len(phage_ids)):
            new_row = self.__load_fasta__(phage_ids[x])
            df_seqs = pd.concat([df_seqs, new_row], ignore_index=True)

        # build a lookup dict
        phage_to_seq = dict(zip(df_seqs["id"], df_seqs["sequence"]))
```

```
        phage_to_desc  = dict(zip(df_seqs["id"], df_seqs["description"]))
        phage_to_len   = dict(zip(df_seqs["id"], df_seqs["length"]))

        rows = []

        for x, pro_row in protein_df.iterrows():
            phage_id = pro_row["Phage_ID"]
            start = int(pro_row["Start"])
            stop = int(pro_row["Stop"])
            strand = pro_row["Strand"]
            prot_id = pro_row.get("Protein_ID")

            genome_seq = phage_to_seq.get(phage_id)
            if genome_seq is None:
                dna_seq = None
            else:
                # Prodigal/GFF: [start, stop] is 1-based inclusive
                start_i = start - 1   # 0-based
                stop_i = stop         # exclusive for Python slicing
                dna_seq = genome_seq[start_i:stop_i]
                if strand == "-":
                    dna_seq = self.reverse_complement(dna_seq)

            rows.append(
                {
                    "Protein_ID": prot_id,
                    "sequence": dna_seq,
                    "description": phage_to_desc.get(phage_id),
                    "length": phage_to_len.get(phage_id),

                }
            )

        df_out = pd.DataFrame(rows)

        return df_out

    def __get_phages_with_no_AMR__(self,phage_AMR_ids,meta_data, n):
        '''Generates a list of non-AMR phage ids'''
        non_AMR_ids = []

        meta_data_AMR_removed = meta_data[~meta_data["Phage_ID"].isin(phage_AMR_ids)].re
        meta_data_AMR_removed_ids = meta_data_AMR_removed["Phage_ID"].tolist()

        non_AMR_ids = random.sample(meta_data_AMR_removed_ids, n)

        return non_AMR_ids


    def __get_phages_with_AMR__(self,phage_AMR_ids,meta_data, n):
        '''generate a list of AMR ids '''

        AMR_ids = []

        meta_data_AMR = meta_data[meta_data["Phage_ID"].isin(phage_AMR_ids)].reset_index
        meta_data_AMR_ids = meta_data_AMR["Phage_ID"].tolist()

        AMR_ids = random.sample(meta_data_AMR_ids, n)
```

```
        return AMR_ids

    def __remove_repeated_cols_df__(self,df):
        ''' Drop duplicate columns from data frame merger'''

        unique_cols = []
        seen = set()

        for col in df.columns:
            # Remove typical merge suffixes: _x, _y, _left, _right
            base = re.sub(r'(_x$|_y$|_left$|_right$)', '', col, flags=re.IGNORECASE)
            key = base.lower()

            if key not in seen:
                seen.add(key)
                unique_cols.append(col)

        return df[unique_cols]

    def __drop_index_columns__(self,df):
    # drop things like 'index', 'index_x', 'index_y', 'level_0', 'Unnamed: 0', etc.
        bad = [c for c in df.columns
            if c.lower().startswith("index")
            or c.lower().startswith("level_0")
            or c.lower().startswith("unnamed")]
        return df.drop(columns=bad, errors="ignore")

    #Functions for creating datafile
    def create_data_file(self, number_of_AMR_seqs = None, Number_of_control_seqs = None,
                         high_quality_only = False, high_med_quality_only = True, save_f
                         AMR_proteins_Only = False):

        '''Main function for creating datasets:

        Inputs:

        number_of_AMR_seqs = None # number of phage AMR sequences in the dataset, defaul
        Number_of_control_seqs = None # number of phage non-AMR sequences, default is 0
        high_quality_only = False #Filter for reads that are high-quality only
        high_med_quality_only = True #filter for reads that are medium and high quality
        save_fname = None #export file name, Also doubles as True for save data, data is
        MR_proteins_Only = True #only AMR protein sequences, overrides number_of_AMR_seq

        outputs:
        saves a .pickle file if save_fname is passed
        initilizes object with data in self.data_df'''


        #Load in all data files
        phage_AMR_df = self.__load_tsv__(self.amr_tsv)
        gff3_df = self.__load_gff3__(self.gff3_file)
        protein_df = self.__load_tsv__(self.protein_tsv)
        meta_df = self.__load_tsv__(self.meta_tsv)


        #get All AMR phage data
        phage_AMR_ids = phage_AMR_df['Phage_id']

        #filter based on completness, Can also filter based on score in gff3 but this se
```

```
        if high_quality_only is True:
            meta_df = meta_df[meta_df["Completeness"] == "High-quality"].reset_index()
        elif high_med_quality_only is True:
            meta_df = meta_df[(meta_df["Completeness"] == "High-quality") | (meta_df["Co

        #get a number of AMR phage ids
        if number_of_AMR_seqs is None:
            number_of_AMR_seqs = len(phage_AMR_ids)
        phage_AMR_ids_pos = self.__get_phages_with_AMR__(phage_AMR_ids,meta_df, number_o

        #get a number of non AMR phage ids
        if Number_of_control_seqs is not None:
            phage_control_ids = self.__get_phages_with_no_AMR__(phage_AMR_ids,meta_df, N
        else:
            phage_control_ids = []

        #Merge control and positive list
        full_list = phage_AMR_ids_pos+phage_control_ids


        #filter all data
        gff3_df = gff3_df.rename(columns={"seqid":"Phage_ID", "start":"Start","end":"Sto
        gff3_df = gff3_df[gff3_df["Phage_ID"].isin(full_list)].reset_index()
        protein_df = protein_df[protein_df["Phage_ID"].isin(full_list)].reset_index()
        meta_df = meta_df[meta_df["Phage_ID"].isin(full_list)].reset_index()

        #Loada fasta files
        seq_df = self.__get_Phage_seqs__(full_list,protein_df=protein_df)
        seq_df = seq_df.rename(columns={"id":"Phage_ID"})

        #drop index cols

        gff3_df=self.__drop_index_columns__(gff3_df)
        protein_df=self.__drop_index_columns__(protein_df)
        meta_df=self.__drop_index_columns__(meta_df)
        seq_df= self.__drop_index_columns__(seq_df)
        #merge dfs
        df_one = meta_df.merge(protein_df, on="Phage_ID", how="left")


        df_two = seq_df.merge(df_one,on="Protein_ID", how="left")


        # numeric coordinates
        for col in ["Start", "Stop"]:
            gff3_df[col] = gff3_df[col].astype("int64")
            df_two[col]  = pd.to_numeric(df_two[col], errors="coerce").astype("Int64")

        # string IDs / strand
        for col in ["Phage_ID", "Strand"]:
            gff3_df[col] = gff3_df[col].astype(str).str.strip()
            df_two[col]  = df_two[col].astype(str).str.strip()


        merge_keys = ["Phage_ID", "Start", "Stop", "Strand"]

        df_all = gff3_df.merge(df_two,on=merge_keys,how="left",validate="one_to_one")

        #Remove duplicate coloumns
```

```
        self.data_df = self.__remove_repeated_cols_df__(df_all)

        #drop garbage cols for reduced file size
        self.data_df = self.data_df.drop(columns=['Function_prediction_source','Protein_
                                                  'Lifestyle','Cluster','Subcluster','so
        #Filter for only AMR proteins
        if AMR_proteins_Only is True:
             self.data_df = self.data_df[self.data_df["Protein_ID"].isin(phage_AMR_df['P

        if save_fname is not None:
            self.savePickle(save_fname,self.data_df)

        return
```

In [2]:

```
'''Example script to organize and merge all of the data for a dataset from PhageScope
For this code to work the following files are required:

    -TemPhD # folder with all of the fasta files
    -TemPhD_antimicrobial_resistance_gene_data.tsv
    -temphd_phage_annotated_protein_meta_data.tsv
    -temphd_phage_meta_data.tsv
    -TemPhD.gff3

Data can be saved and loaded as .pickle files. There is also an option to export it to .
'''

#%% Create Data object
# Folder containing all the TSV/GFF and the 'TemPhD' FASTA folder
folder_path = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
db_name = "TemPhD"

data_obj = DataFormat(folder_path, db_name)

#%% Generate Data
# Parameters for dataset creation
number_of_AMR_seqs = 10         # number of AMR sequences to include
Number_of_control_seqs = None   # default: 0 non-AMR sequences
high_quality_only = False       # filter only high-quality reads
high_med_quality_only = True    # filter medium & high quality reads
save_fname = None               # None = no automatic save
MR_proteins_Only = True         # only AMR protein sequences

# Call the main function
data_obj.create_data_file(
    number_of_AMR_seqs=number_of_AMR_seqs,
    Number_of_control_seqs=Number_of_control_seqs,
    high_quality_only=high_quality_only,
    high_med_quality_only=high_med_quality_only,
    save_fname=save_fname,
    AMR_proteins_Only=MR_proteins_Only
)

#%% Optional: Export merged data to CSV
# data_obj.export_csv("merged_phagescope_data.csv", data_obj.data_df)
```

In [4]:

```
import numpy as np
import pandas as pd
```

```python
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
```

```
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ==========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ==========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=1000, step=250, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
```

```
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
```

```
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
```

```
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
```

```
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
```

```
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)

    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)
```

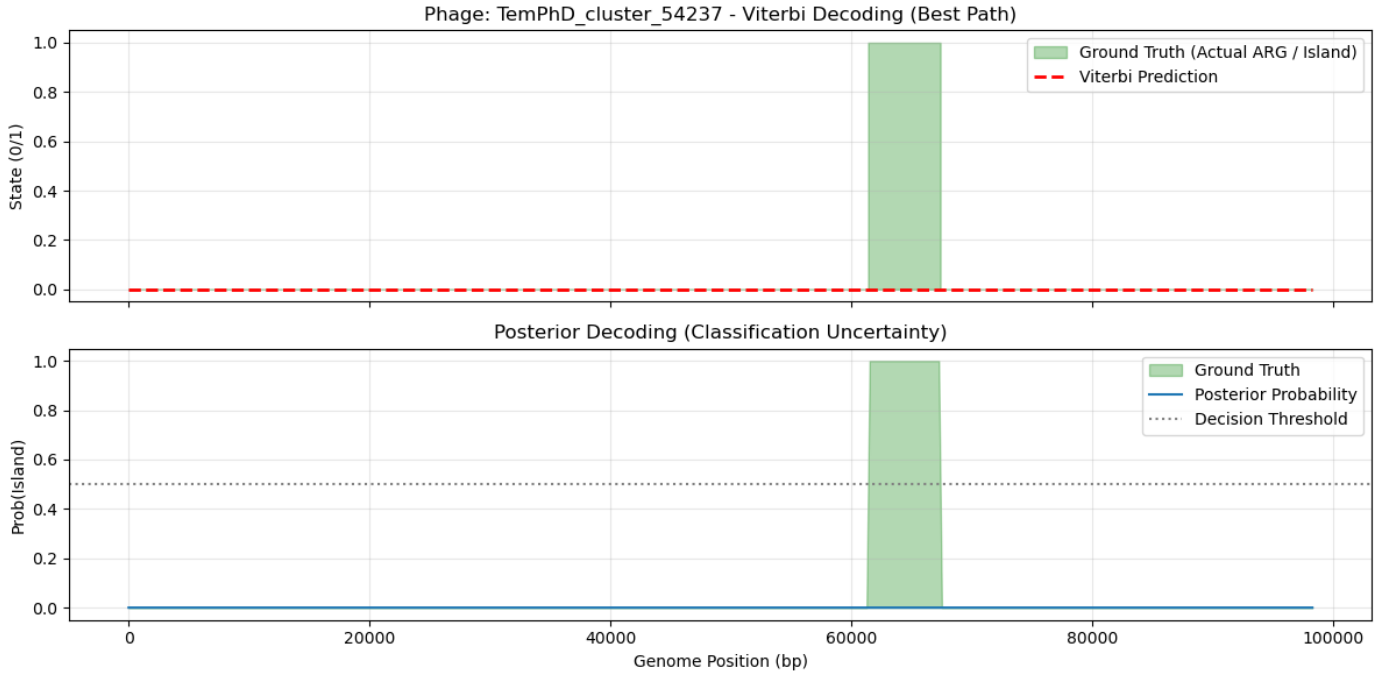
```
    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-----------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
```

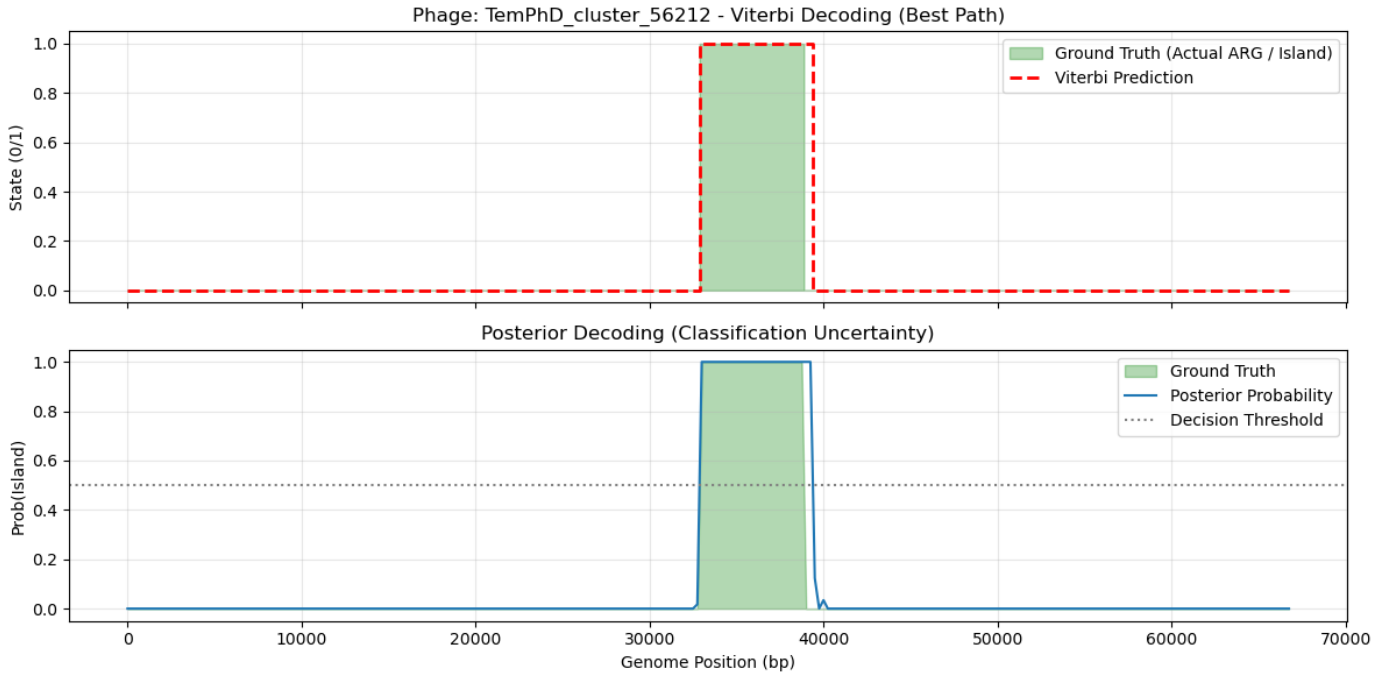
```
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }
```

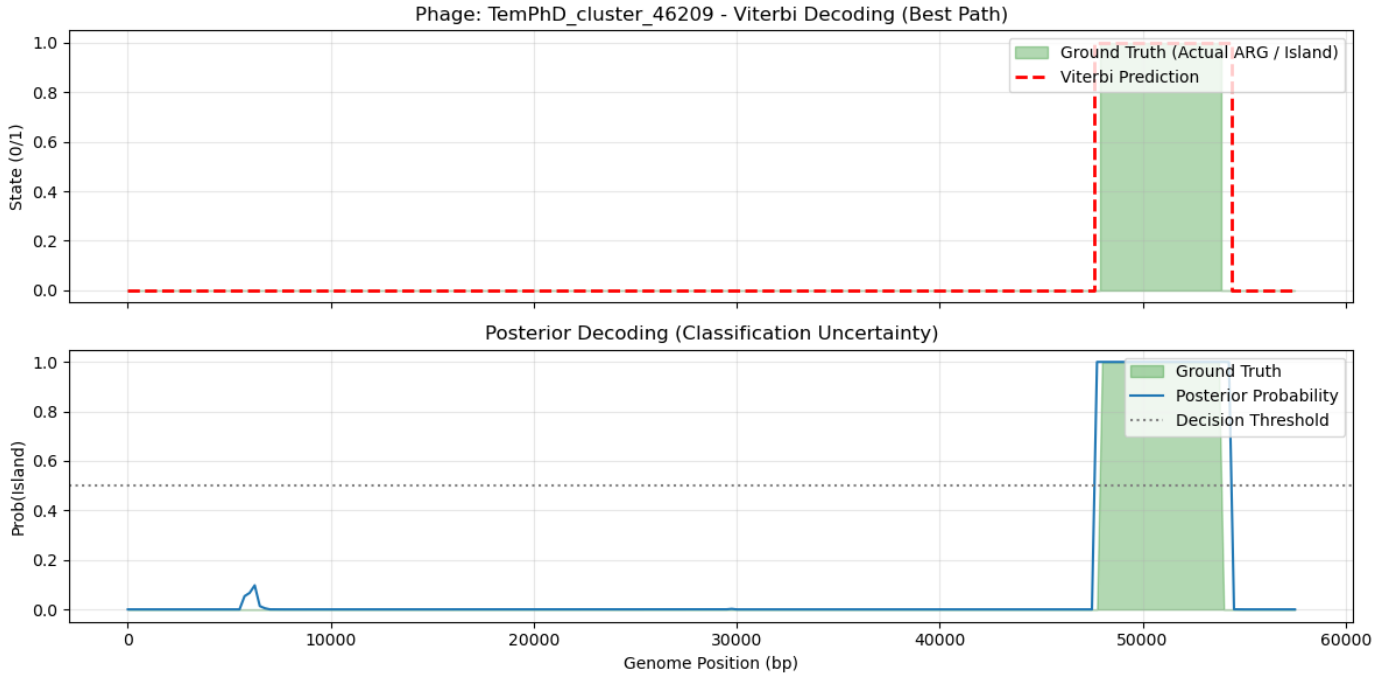
```python
# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=1000,
    step=250,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 35
Examples (first 10):
- TemPhD_cluster_46242: [{'start': 1250, 'end': 8000, 'length': 6750, 'arg_count': 1, 'm
obile_count': 2, 'gc_shift': 0.032191993632864235}] | #args=2 mobile_hits=4 genome_GC=0.
394
- TemPhD_cluster_39211: [{'start': 37000, 'end': 43750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.0033462167099317375}] | #args=1 mobile_hits=3 genome_GC
=0.367
- TemPhD_cluster_22243: [{'start': 37000, 'end': 43750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003123835229784877}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_56217: [{'start': 33000, 'end': 39750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.0257332710173091}] | #args=1 mobile_hits=4 genome_GC=0.
345
- TemPhD_cluster_22244: [{'start': 37000, 'end': 43750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003219978290659453}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_46209: [{'start': 48000, 'end': 54750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.015212533920434201}] | #args=1 mobile_hits=5 genome_GC=
0.354
- TemPhD_cluster_22237: [{'start': 37000, 'end': 43750, 'length': 6750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003155882916743069}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_54409: [{'start': 27250, 'end': 34000, 'length': 6750, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.000777157353978819}] | #args=1 mobile_hits=4 genome_GC=
0.479
- TemPhD_cluster_54261: [{'start': 37500, 'end': 45000, 'length': 7500, 'arg_count': 2,
'mobile_count': 2, 'gc_shift': 0.05513425312915887}] | #args=3 mobile_hits=6 genome_GC=
0.359
- TemPhD_cluster_55771: [{'start': 61250, 'end': 70500, 'length': 9250, 'arg_count': 2,
'mobile_count': 3, 'gc_shift': 0.043768527477040686}] | #args=2 mobile_hits=7 genome_GC=
0.398

[CHECKPOINT] LASSO selected 202 features.
```

```
[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.7168
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      1.00      1.00     15625
      Island       0.81      0.80      0.81       168

    accuracy                           1.00     15793
   macro avg       0.90      0.90      0.90     15793
weighted avg       1.00      1.00      1.00     15793

Confusion Matrix:
[[15593    32]
 [   33   135]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```








```
Predicted islands after merging (count = 7). Example:
              Phage_ID  pred_island_start  pred_island_end  length
0  TemPhD_cluster_46242               1750             6250    4500
1  TemPhD_cluster_56564              49000            52750    3750
2  TemPhD_cluster_46209              47750            55250    7500
3  TemPhD_cluster_56212              33000            40250    7250
4  TemPhD_cluster_22242              36750            44250    7500

PIPELINE COMPLETE. Returning results (no CSVs written).
```

In [5]:

```
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
```

```python
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
```

```
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ==========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ==========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=1000, step=500, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
```

```
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("------------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")
```

```
    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1
```

```
            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
```

```
                'mobile_count': mobile_count_in_region,
                'gc_shift': gc_shift
            })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
```

```
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)

    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
```

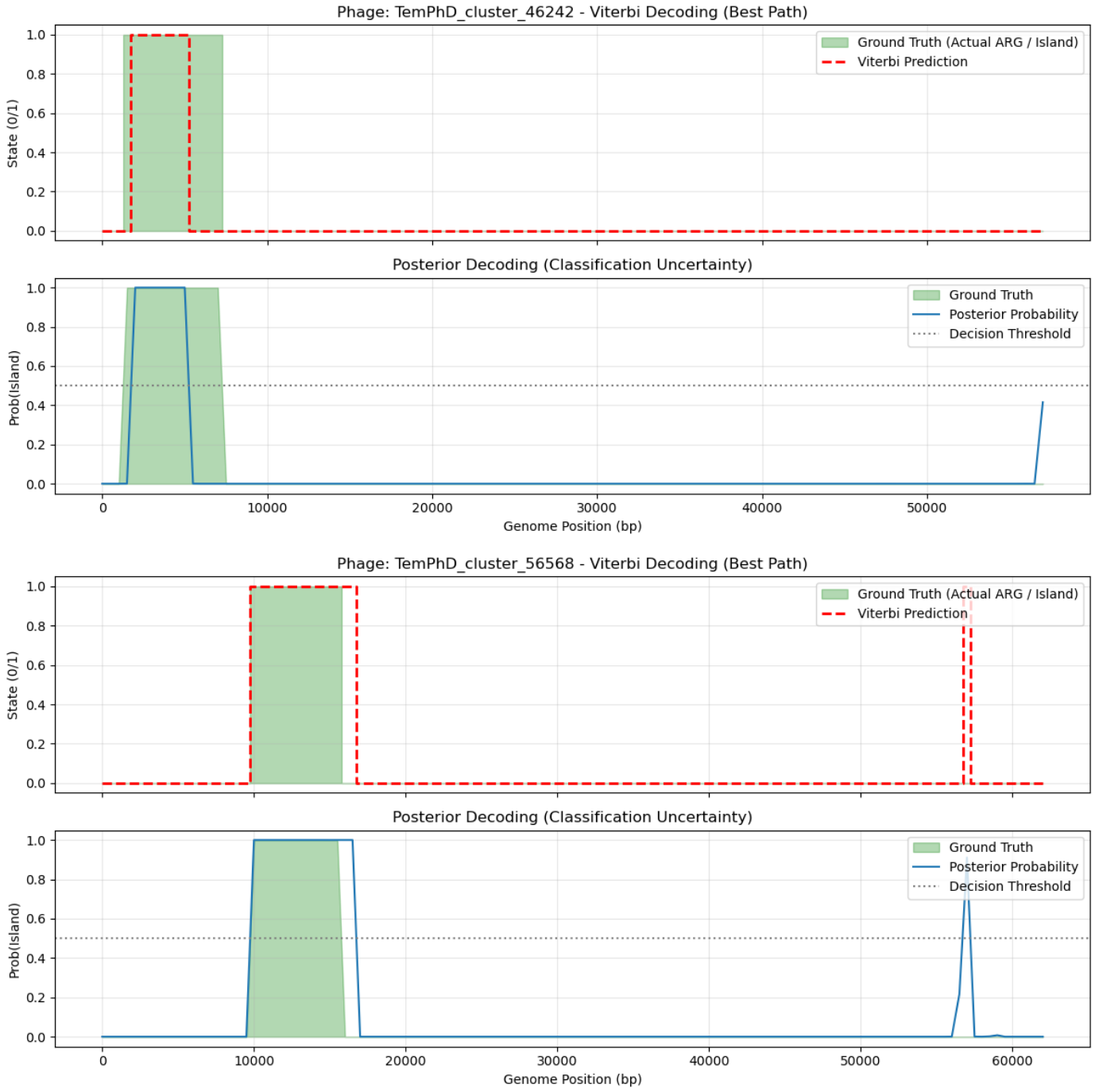
```
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")
```

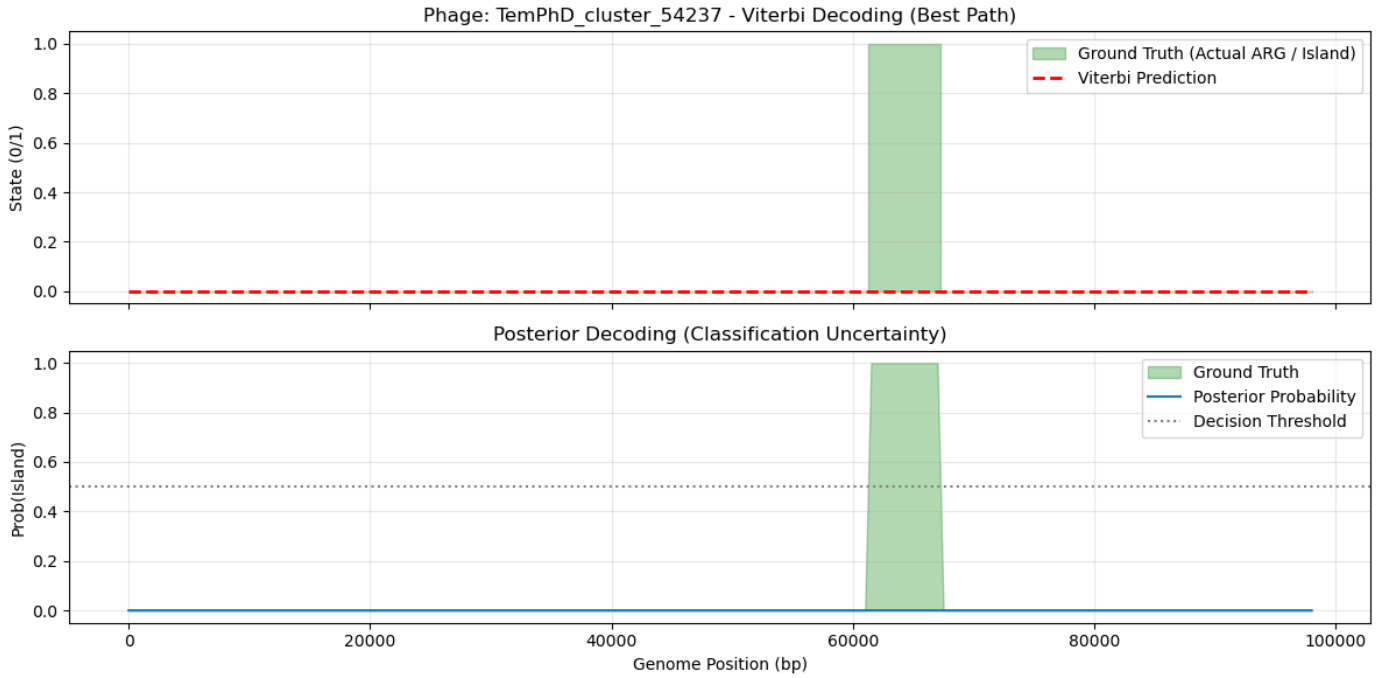
```
    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=1000,
```

```
    step=500,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 14
Examples (first 10):
- TemPhD_cluster_46242: [{'start': 1500, 'end': 8000, 'length': 6500, 'arg_count': 1, 'm
obile_count': 2, 'gc_shift': 0.03287575431662493}] | #args=2 mobile_hits=4 genome_GC=0.3
94
- TemPhD_cluster_54409: [{'start': 27500, 'end': 34000, 'length': 6500, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.0003168597400383111}] | #args=1 mobile_hits=4 genome_GC
=0.479
- TemPhD_cluster_54261: [{'start': 37500, 'end': 45000, 'length': 7500, 'arg_count': 2,
'mobile_count': 2, 'gc_shift': 0.05513425312915887}] | #args=3 mobile_hits=6 genome_GC=
0.359
- TemPhD_cluster_55771: [{'start': 61500, 'end': 70500, 'length': 9000, 'arg_count': 2,
'mobile_count': 3, 'gc_shift': 0.04482858753710073}] | #args=2 mobile_hits=7 genome_GC=
0.398
- TemPhD_cluster_45788: [{'start': 61500, 'end': 70500, 'length': 9000, 'arg_count': 2,
'mobile_count': 3, 'gc_shift': 0.04458295402889206}] | #args=2 mobile_hits=7 genome_GC=
0.398
- TemPhD_cluster_55772: [{'start': 61500, 'end': 70500, 'length': 9000, 'arg_count': 2,
'mobile_count': 3, 'gc_shift': 0.04490285028582558}] | #args=2 mobile_hits=7 genome_GC=
0.398
- TemPhD_cluster_56167: [{'start': 7000, 'end': 13500, 'length': 6500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.024757012265705203}] | #args=2 mobile_hits=4 genome_GC=
0.356
- TemPhD_cluster_45961: [{'start': 11500, 'end': 18000, 'length': 6500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.01686148347571015}] | #args=1 mobile_hits=4 genome_GC=
0.345
- TemPhD_cluster_58001: [{'start': 46500, 'end': 53000, 'length': 6500, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.01593907074472445}] | #args=3 mobile_hits=3 genome_GC=
0.399
- TemPhD_cluster_55462: [{'start': 46000, 'end': 52500, 'length': 6500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.027468087234893968}] | #args=2 mobile_hits=4 genome_GC=
0.299

[CHECKPOINT] LASSO selected 144 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.7563
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED
```

```
=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      0.99      0.99      7879
      Island       0.20      0.53      0.29        36

    accuracy                           0.99      7915
   macro avg       0.60      0.76      0.64      7915
weighted avg       0.99      0.99      0.99      7915

Confusion Matrix:
[[7802   77]
 [  17   19]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```






```
Predicted islands after merging (count = 7). Example:
               Phage_ID  pred_island_start  pred_island_end  length
0  TemPhD_cluster_46242               2000             6000    4000
1  TemPhD_cluster_56564              49000            52500    3500
2  TemPhD_cluster_46209              48000            55500    7500
3  TemPhD_cluster_56212              33000            40500    7500
4  TemPhD_cluster_22242              37000            45000    8000

PIPELINE COMPLETE. Returning results (no CSVs written).
```

In [6]:

```python
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
```

```
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=500, step=250, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
```

```
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
```

```
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
```

```
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
```

```
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)
```

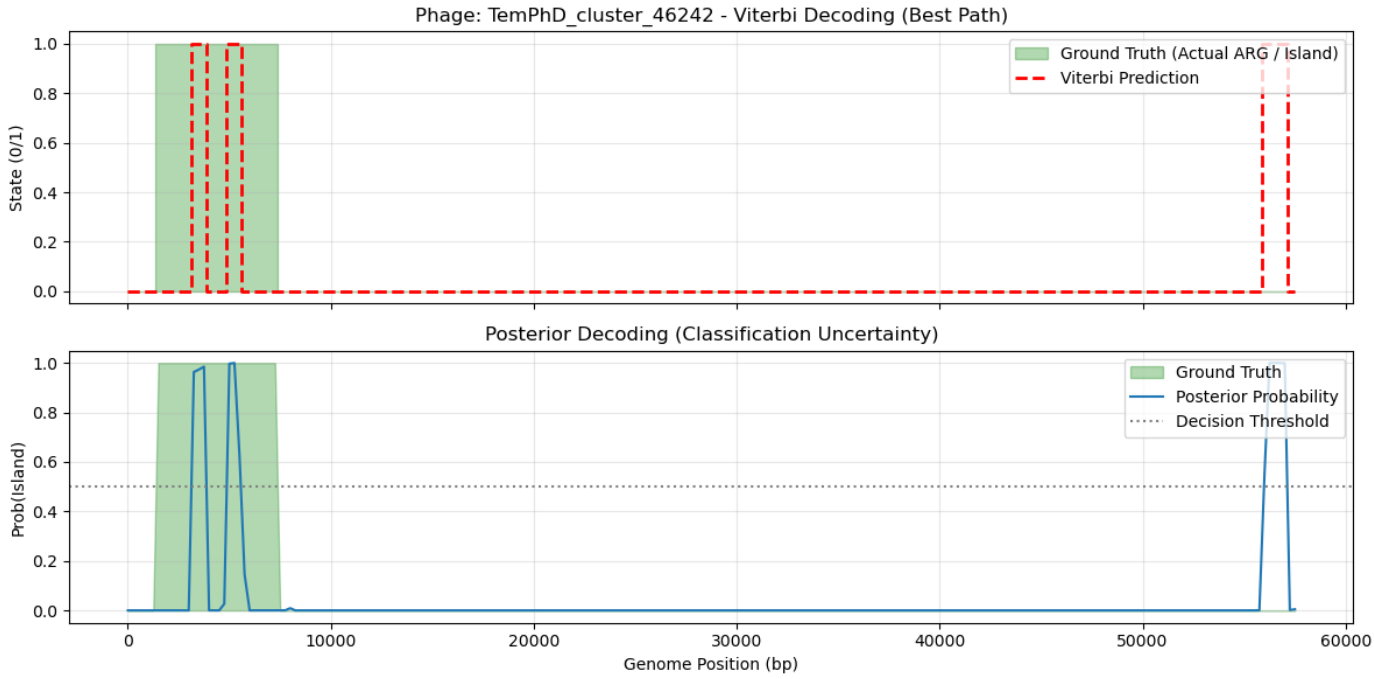
```
    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
```

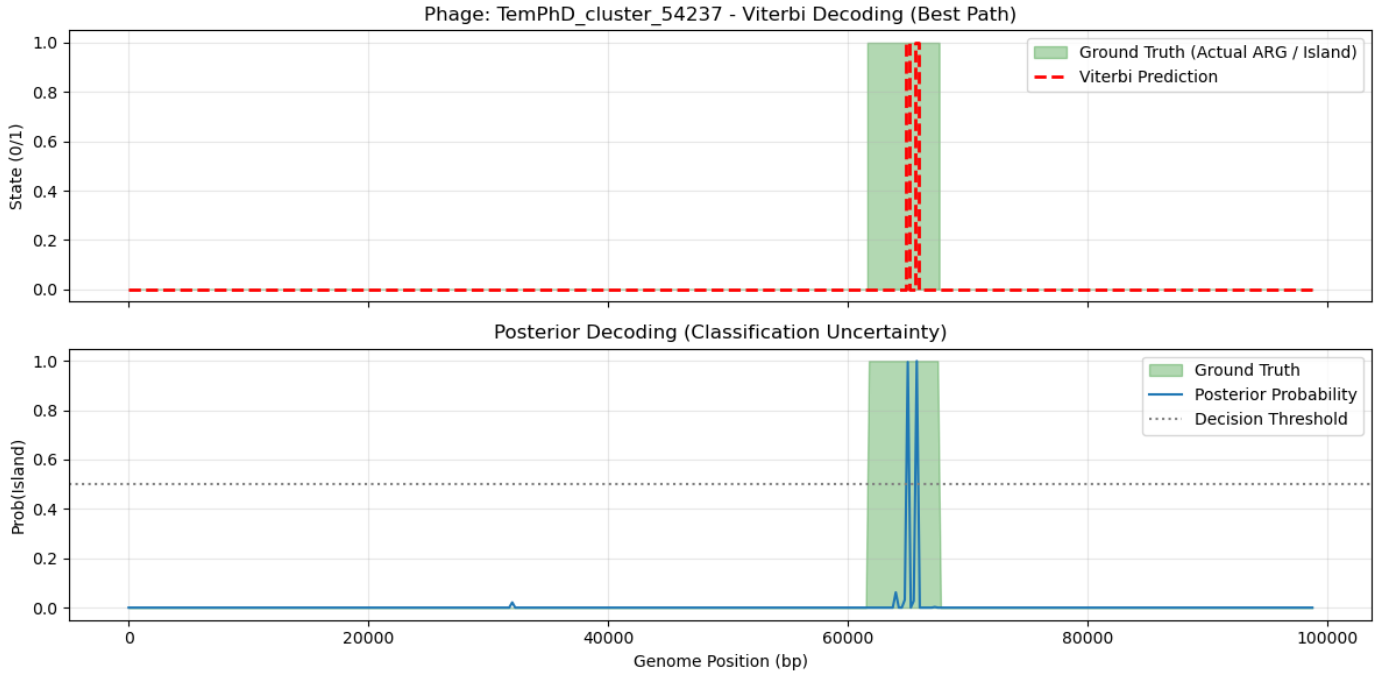
```
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
```

```
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=500,
    step=250,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 10
Examples (first 10):
```

```
- TemPhD_cluster_46242: [{'start': 1500, 'end': 7750, 'length': 6250, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.035694215855086475}] | #args=2 mobile_hits=4 genome_GC=0.394
- TemPhD_cluster_54409: [{'start': 27500, 'end': 33750, 'length': 6250, 'arg_count': 1, 'mobile_count': 3, 'gc_shift': 0.0013816017984232265}] | #args=1 mobile_hits=4 genome_GC=0.479
- TemPhD_cluster_54261: [{'start': 37750, 'end': 44750, 'length': 7000, 'arg_count': 2, 'mobile_count': 2, 'gc_shift': 0.05368663408153984}] | #args=3 mobile_hits=6 genome_GC=0.359
- TemPhD_cluster_55771: [{'start': 61500, 'end': 70250, 'length': 8750, 'arg_count': 2, 'mobile_count': 3, 'gc_shift': 0.04640636531487852}] | #args=2 mobile_hits=7 genome_GC=0.398
- TemPhD_cluster_45788: [{'start': 61500, 'end': 70250, 'length': 8750, 'arg_count': 2, 'mobile_count': 3, 'gc_shift': 0.046836922282860294}] | #args=2 mobile_hits=7 genome_GC=0.398
- TemPhD_cluster_55772: [{'start': 61500, 'end': 70250, 'length': 8750, 'arg_count': 2, 'mobile_count': 3, 'gc_shift': 0.0465949137778891}] | #args=2 mobile_hits=7 genome_GC=0.398
- TemPhD_cluster_56167: [{'start': 7250, 'end': 13500, 'length': 6250, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.022855473804166748}] | #args=2 mobile_hits=4 genome_GC=0.356
- TemPhD_cluster_58001: [{'start': 46750, 'end': 53000, 'length': 6250, 'arg_count': 1, 'mobile_count': 3, 'gc_shift': 0.014603686129339821}] | #args=3 mobile_hits=3 genome_GC=0.399
- TemPhD_cluster_55462: [{'start': 46250, 'end': 52500, 'length': 6250, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.030748087234893973}] | #args=2 mobile_hits=4 genome_GC=0.299
- TemPhD_cluster_54237: [{'start': 61750, 'end': 68000, 'length': 6250, 'arg_count': 1, 'mobile_count': 3, 'gc_shift': 0.0006337077927310864}] | #args=1 mobile_hits=4 genome_GC=0.479

[CHECKPOINT] LASSO selected 106 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.7747
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      0.99      0.99     15877
      Island       0.04      0.17      0.06        48

    accuracy                           0.98     15925
   macro avg       0.52      0.58      0.52     15925
weighted avg       0.99      0.98      0.99     15925

Confusion Matrix:
[[15658   219]
 [   40     8]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```





Predicted islands after merging (count = 7). Example:

	Phage_ID	pred_island_start	pred_island_end	length
0	TemPhD_cluster_56564	49000	52500	3500
1	TemPhD_cluster_46209	48250	52750	4500
2	TemPhD_cluster_39990	10250	12250	2000
3	TemPhD_cluster_56212	33250	38000	4750
4	TemPhD_cluster_22242	37250	42000	4750

PIPELINE COMPLETE. Returning results (no CSVs written).

In [10]:

```
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
```

```python
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
```

```
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ==========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ==========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=1000, step=100, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
```

```
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("------------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")
```

```
    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1
```

```
            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
```

```
                'mobile_count': mobile_count_in_region,
                'gc_shift': gc_shift
            })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
```

```
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)

    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
```

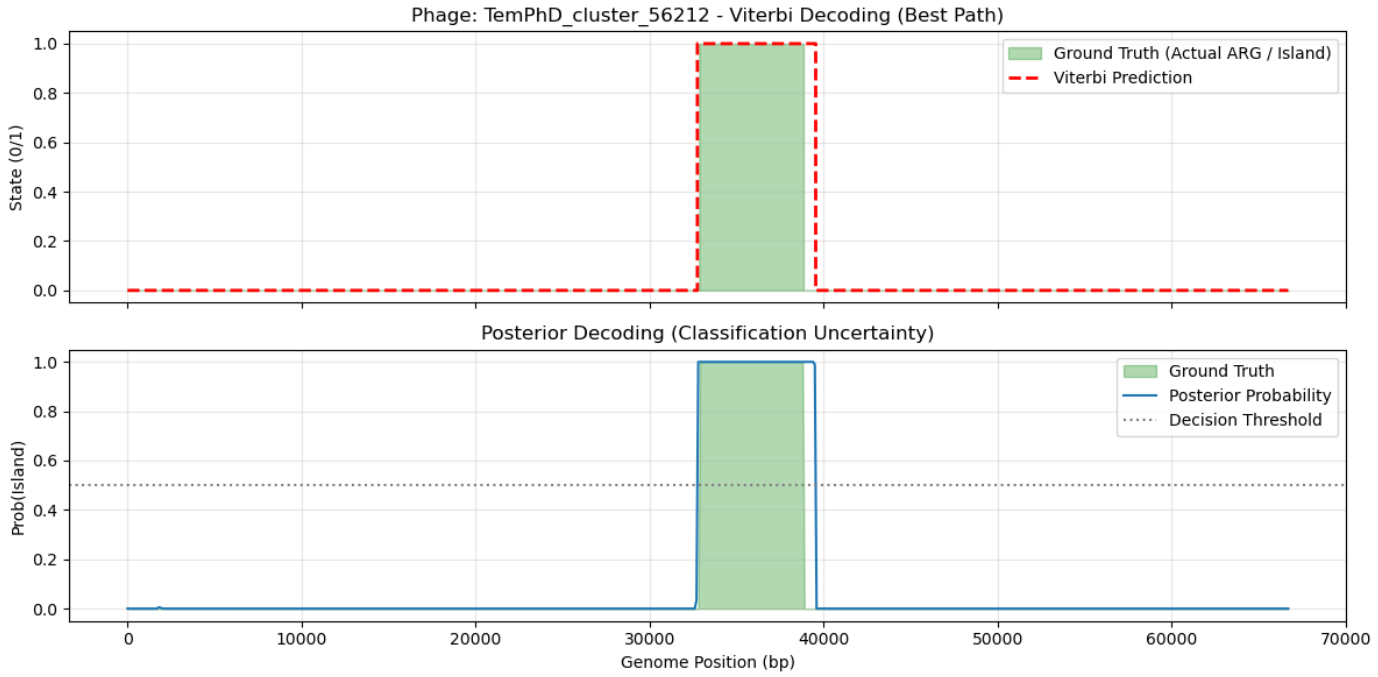
```
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")
```

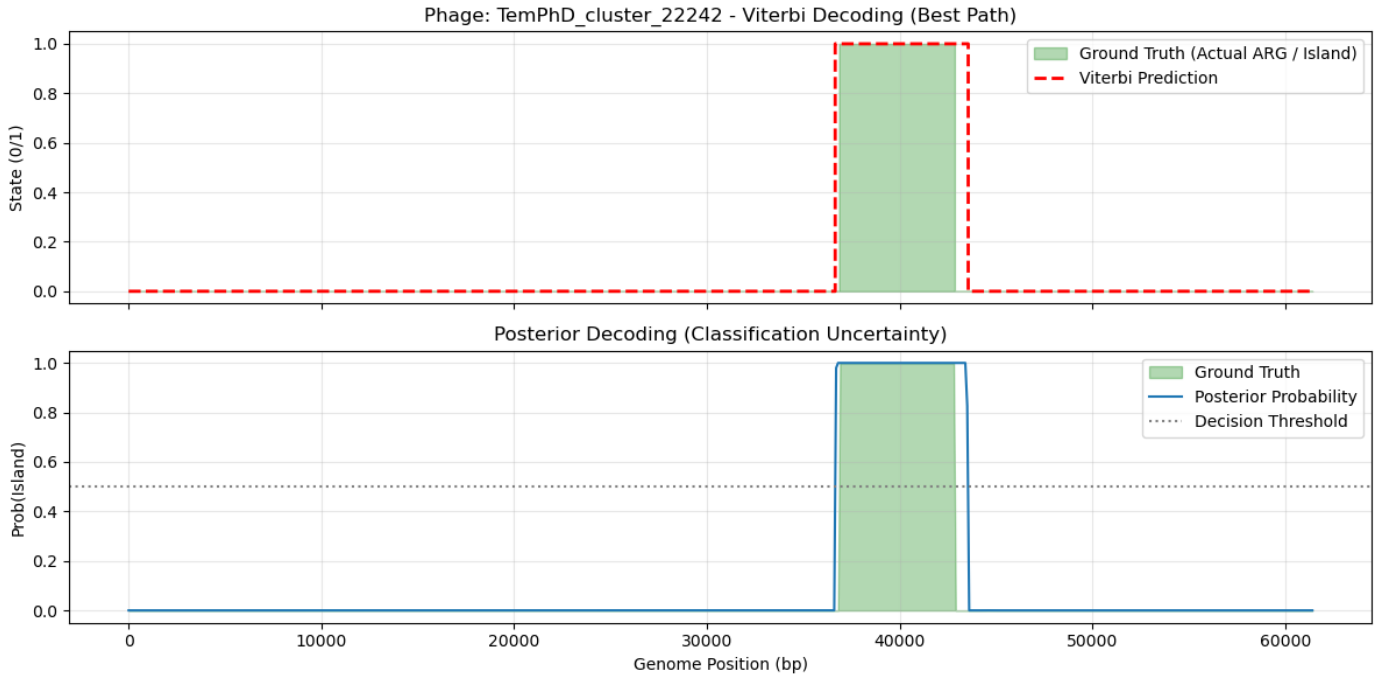
```
    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=1000,
```

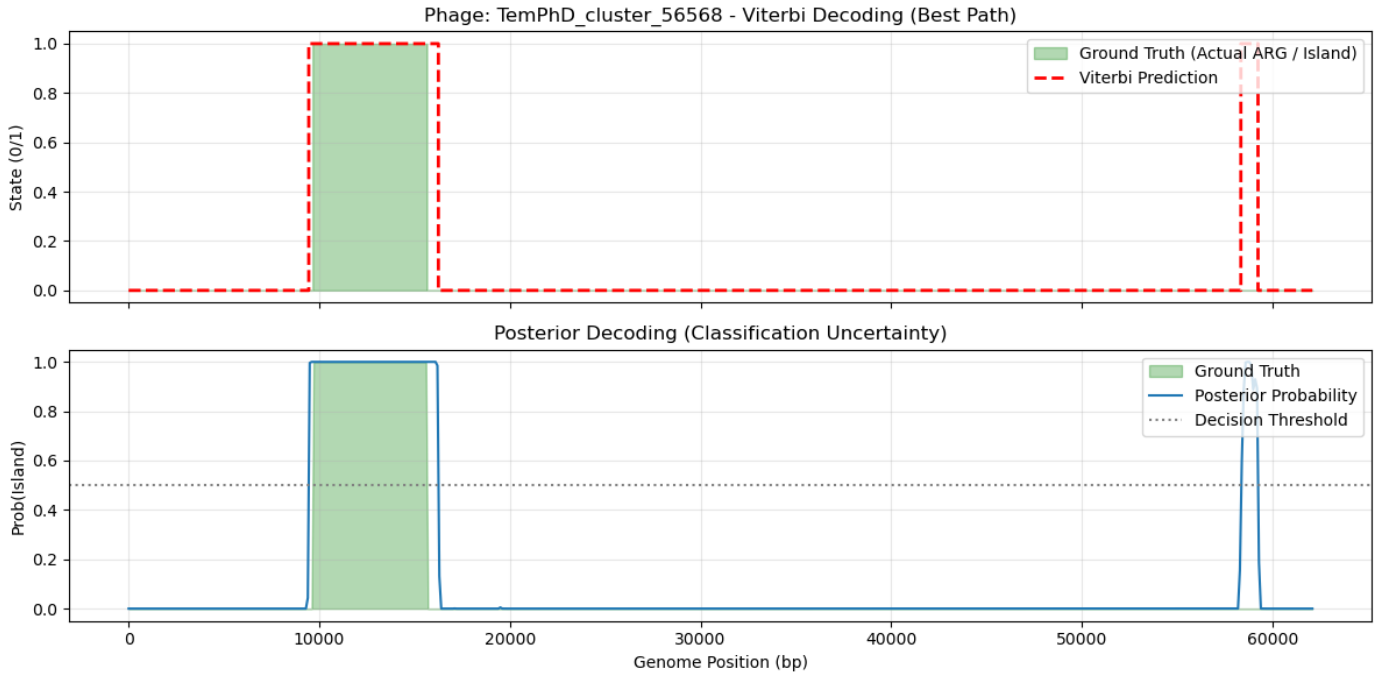
```
    step=100,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 36
Examples (first 10):
- TemPhD_cluster_46242: [{'start': 1100, 'end': 8000, 'length': 6900, 'arg_count': 1, 'm
obile_count': 2, 'gc_shift': 0.03238523034784008}] | #args=2 mobile_hits=4 genome_GC=0.3
94
- TemPhD_cluster_39211: [{'start': 36900, 'end': 43800, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003694042796888286}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_22243: [{'start': 36900, 'end': 43800, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.0037647370011213988}] | #args=1 mobile_hits=3 genome_GC
=0.367
- TemPhD_cluster_56217: [{'start': 32900, 'end': 39800, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.026367731564813146}] | #args=1 mobile_hits=4 genome_GC=
0.345
- TemPhD_cluster_22244: [{'start': 36900, 'end': 43800, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003860880061995975}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_46209: [{'start': 47800, 'end': 54700, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.017473403485651573}] | #args=1 mobile_hits=5 genome_GC=
0.354
- TemPhD_cluster_22237: [{'start': 36900, 'end': 43800, 'length': 6900, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.003796784688079591}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_54409: [{'start': 27100, 'end': 34000, 'length': 6900, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.0022360945520464304}] | #args=1 mobile_hits=4 genome_GC
=0.479
- TemPhD_cluster_54261: [{'start': 37500, 'end': 45200, 'length': 7700, 'arg_count': 2,
'mobile_count': 2, 'gc_shift': 0.05412819252309825}] | #args=3 mobile_hits=6 genome_GC=
0.359
- TemPhD_cluster_55771: [{'start': 61200, 'end': 70600, 'length': 9400, 'arg_count': 2,
'mobile_count': 3, 'gc_shift': 0.042925514251048735}] | #args=2 mobile_hits=7 genome_GC=
0.398

[CHECKPOINT] LASSO selected 199 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.7176
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED
```

```
=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      1.00      1.00     39018
      Island       0.65      0.81      0.72       420

    accuracy                           0.99     39438
   macro avg       0.82      0.90      0.86     39438
weighted avg       0.99      0.99      0.99     39438

Confusion Matrix:
[[38835   183]
 [   79   341]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```







Predicted islands after merging (count = 7). Example:

	Phage_ID	pred_island_start	pred_island_end	length
0	TemPhD_cluster_46242	1800	6400	4600
1	TemPhD_cluster_56564	49000	52800	3800
2	TemPhD_cluster_46209	47600	55400	7800
3	TemPhD_cluster_56212	32800	40500	7700
4	TemPhD_cluster_22242	36700	44500	7800

PIPELINE COMPLETE. Returning results (no CSVs written).

In [12]:

```
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
```

```
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=2000, step=250, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
```

```
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
```

```
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
```

```
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
```

```
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)
```

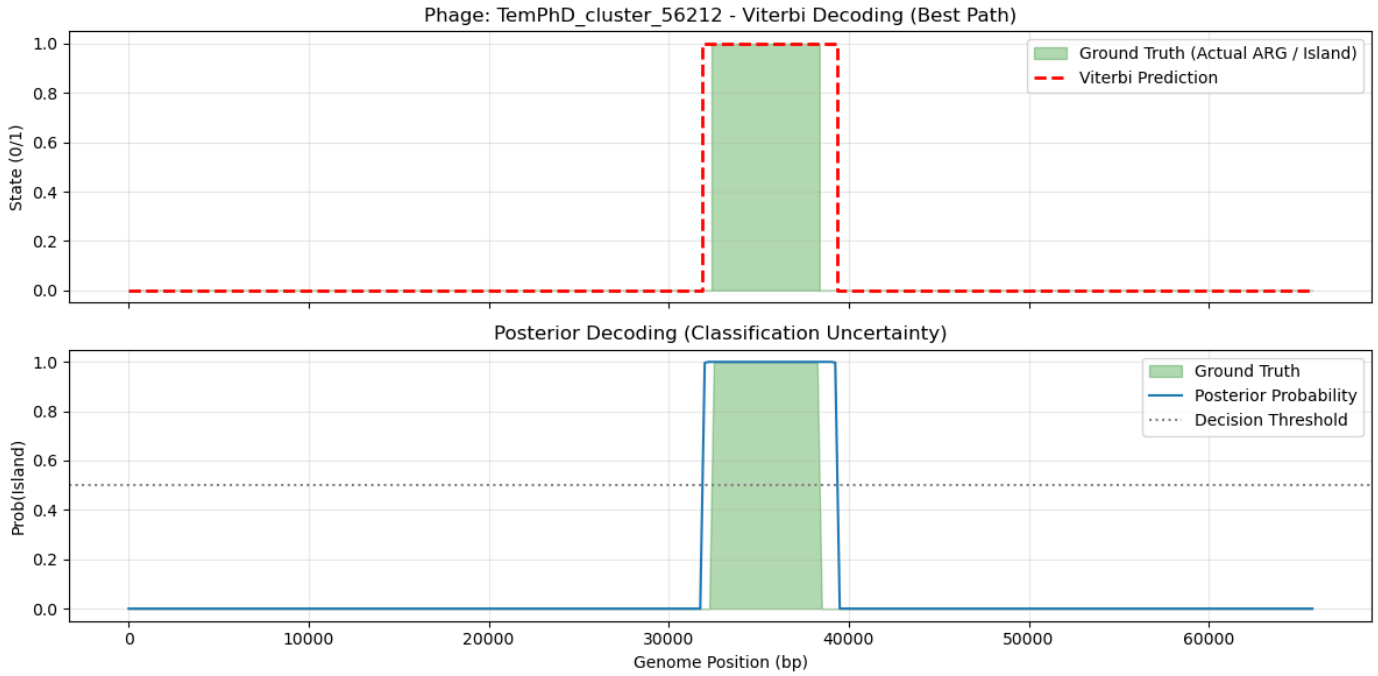
```
    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
```

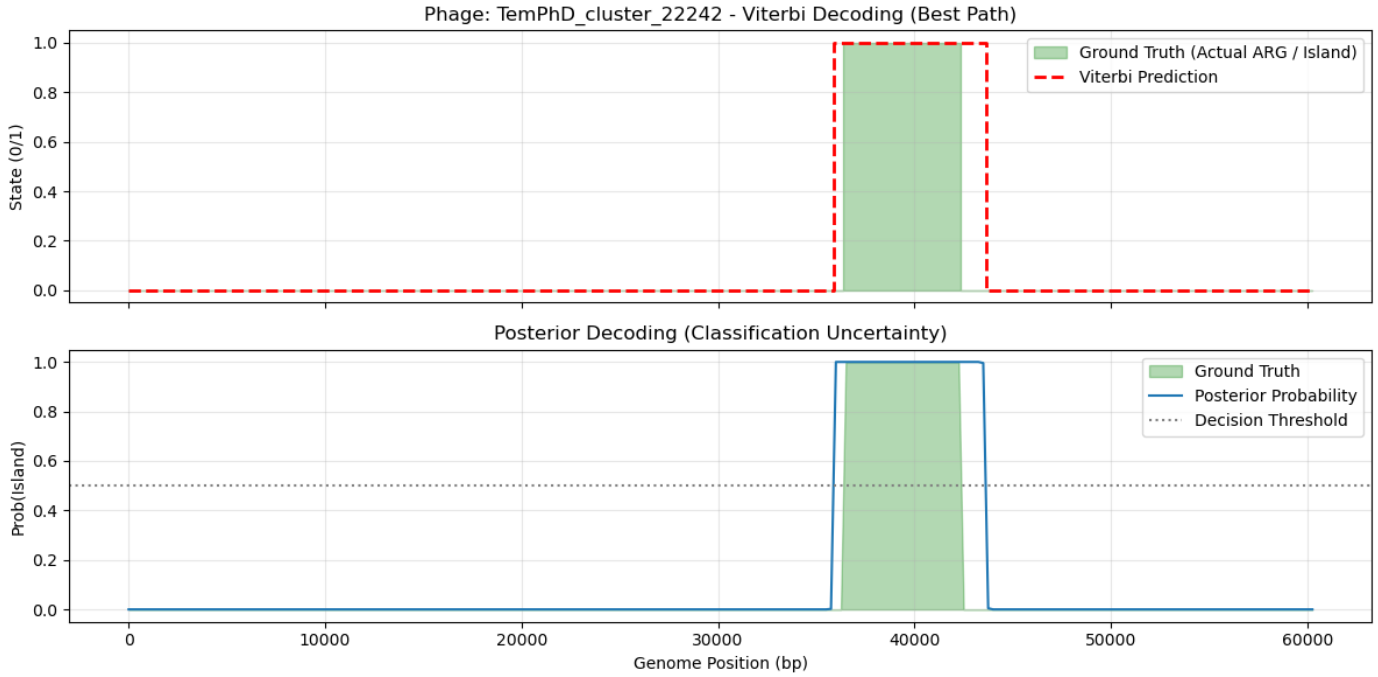
```
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
```

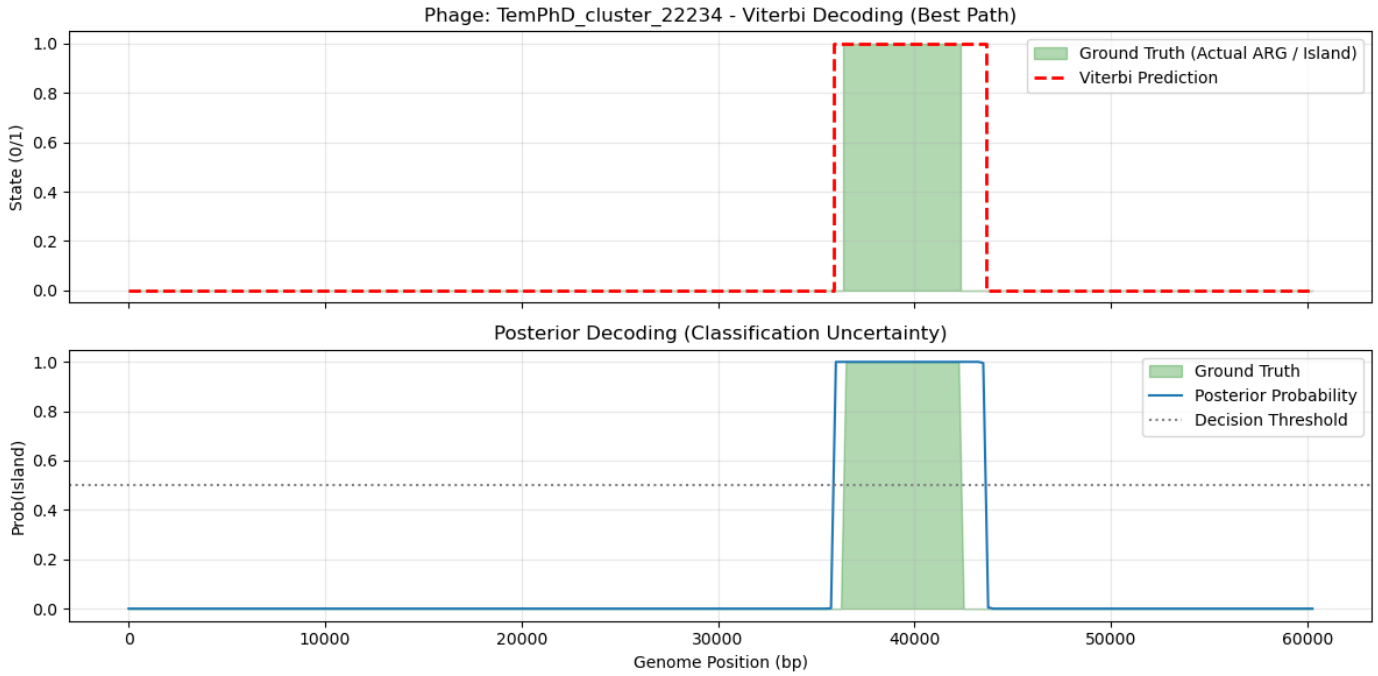
```
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=2000,
    step=250,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 72
Examples (first 10):
```

```
- TemPhD_cluster_56174: [{'start': 47000, 'end': 54750, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.038304219167846376}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_56171: [{'start': 47000, 'end': 54750, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.037911393044002484}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_46242: [{'start': 750, 'end': 8500, 'length': 7750, 'arg_count': 1, 'mo
bile_count': 3, 'gc_shift': 0.03183357069379611}] | #args=2 mobile_hits=4 genome_GC=0.39
4
- TemPhD_cluster_25241: [{'start': 47000, 'end': 54750, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.037702255752617375}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_39211: [{'start': 36500, 'end': 44250, 'length': 7750, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.006486001656168305}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_23154: [{'start': 47000, 'end': 54750, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.038208881605327205}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_22243: [{'start': 36500, 'end': 44250, 'length': 7750, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.006411768324169564 4}] | #args=1 mobile_hits=3 genome_GC
=0.367
- TemPhD_cluster_23148: [{'start': 47000, 'end': 54750, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03822350533671853}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_56217: [{'start': 32500, 'end': 40250, 'length': 7750, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.028853940073462037}] | #args=1 mobile_hits=4 genome_GC=
0.345
- TemPhD_cluster_23152: [{'start': 13750, 'end': 21500, 'length': 7750, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.038725010637585267}] | #args=1 mobile_hits=5 genome_GC=
0.304

[CHECKPOINT] LASSO selected 183 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.8031
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      1.00      1.00     15326
      Island       0.74      0.79      0.76       203

    accuracy                           0.99     15529
   macro avg       0.87      0.89      0.88     15529
weighted avg       0.99      0.99      0.99     15529

Confusion Matrix:
[[15268    58]
 [   42   161]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```







Predicted islands after merging (count = 10). Example:

	Phage_ID	pred_island_start	pred_island_end	length
0	TemPhD_cluster_46242	1750	7250	5500
1	TemPhD_cluster_56564	49000	53500	4500
2	TemPhD_cluster_46209	46750	56500	9750
3	TemPhD_cluster_56212	32000	41250	9250
4	TemPhD_cluster_45874	46000	55500	9500

PIPELINE COMPLETE. Returning results (no CSVs written).

In [13]:

```
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
```

```
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=2000, step=500, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
```

```
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
```

```
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
```

```
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
```

```
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)
```

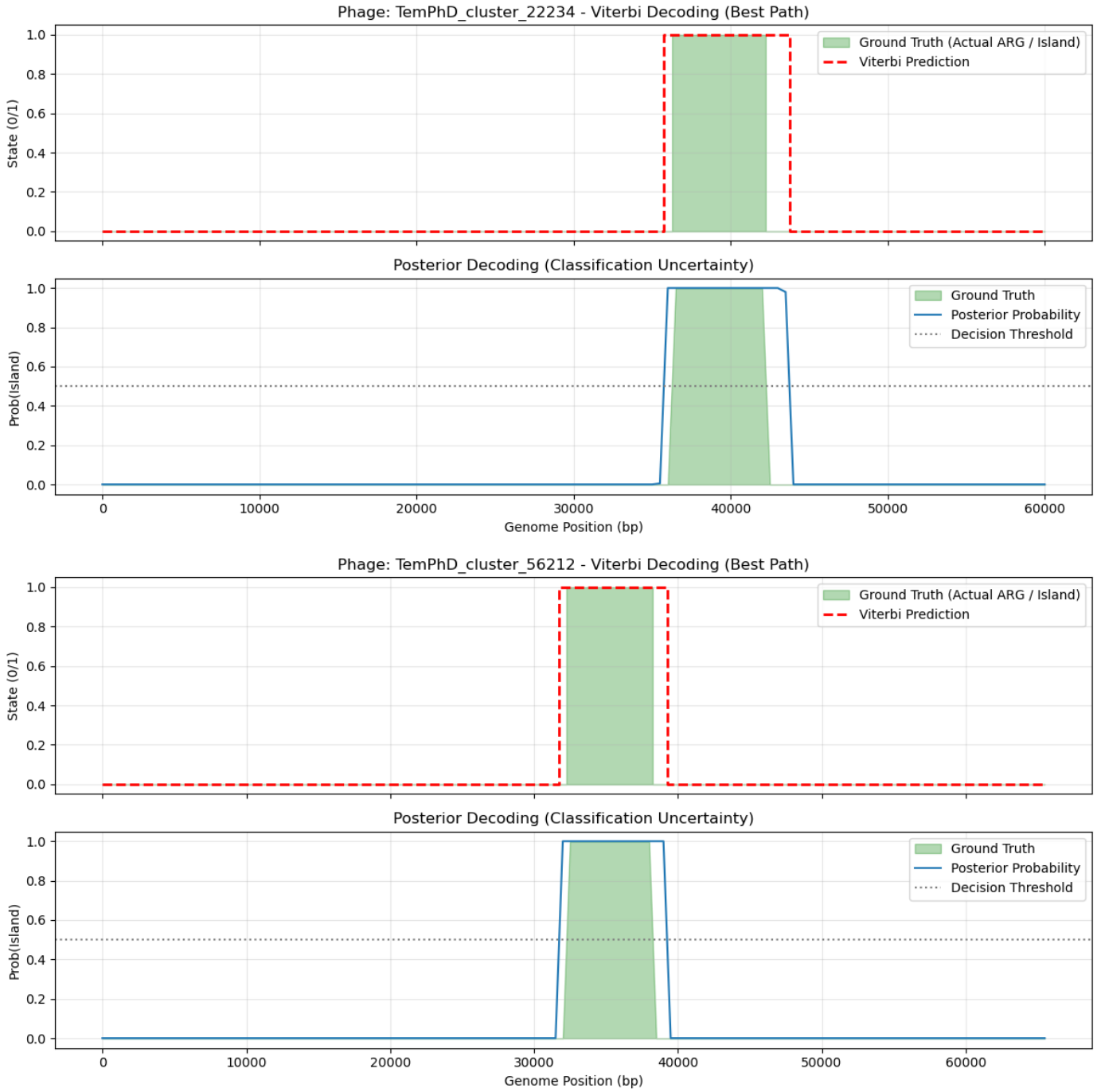
```
    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
```

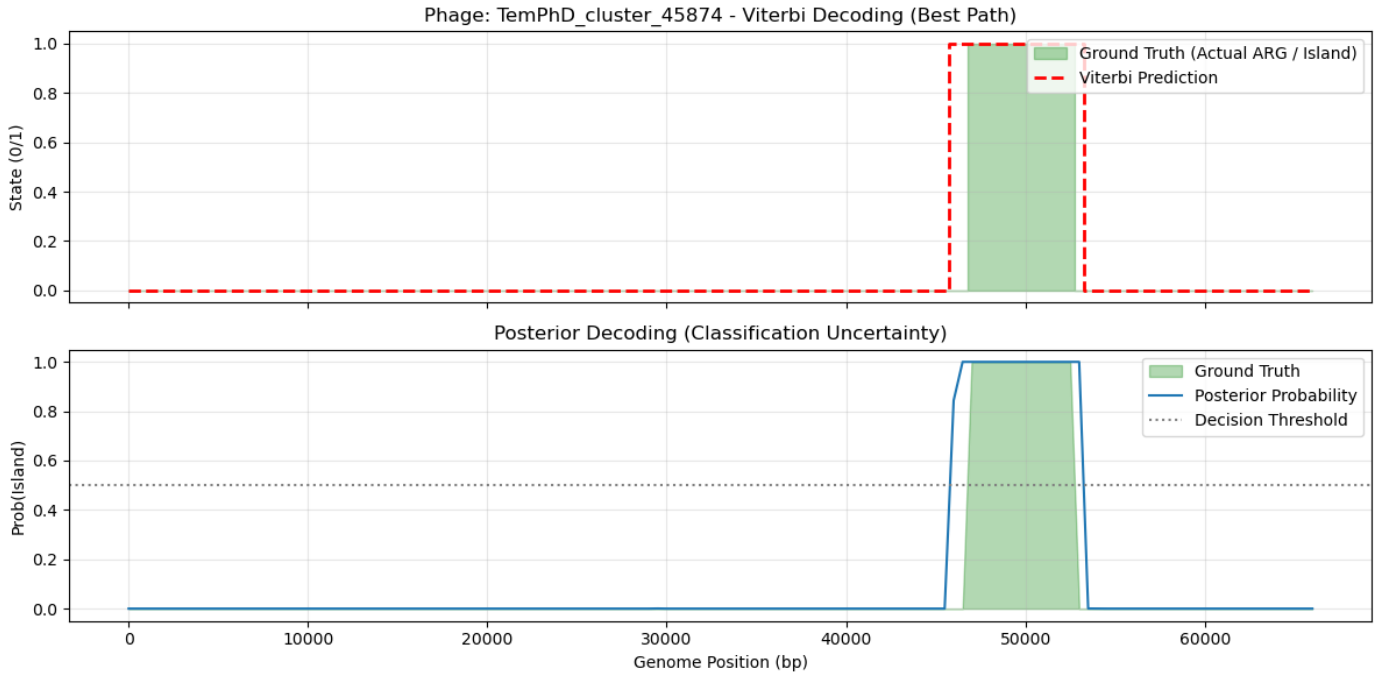
```
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
```

```
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=2000,
    step=500,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 72
Examples (first 10):
```

```
- TemPhD_cluster_56174: [{'start': 47000, 'end': 54500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.04011067078074959}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_56171: [{'start': 47000, 'end': 54500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03970494143109926}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_46242: [{'start': 1000, 'end': 8500, 'length': 7500, 'arg_count': 1, 'm
obile_count': 3, 'gc_shift': 0.03161421585508645}] | #args=2 mobile_hits=4 genome_GC=0.3
94
- TemPhD_cluster_25241: [{'start': 47000, 'end': 54500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03935386865584323}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_39211: [{'start': 36500, 'end': 44000, 'length': 7500, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.005879550043265069}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_23154: [{'start': 47000, 'end': 54500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.04001103214296159}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_22243: [{'start': 36500, 'end': 44000, 'length': 7500, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.005805316711266328}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_23148: [{'start': 47000, 'end': 54500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.04002565587435292}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_56217: [{'start': 32500, 'end': 40000, 'length': 7500, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.02825178953582763}] | #args=1 mobile_hits=4 genome_GC=
0.345
- TemPhD_cluster_23152: [{'start': 14000, 'end': 21500, 'length': 7500, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.0404110321429616}] | #args=1 mobile_hits=5 genome_GC=0.
304

[CHECKPOINT] LASSO selected 193 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.7949
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      1.00      1.00      7681
      Island       0.74      0.76      0.75       102

    accuracy                           0.99      7783
   macro avg       0.87      0.88      0.87      7783
weighted avg       0.99      0.99      0.99      7783

Confusion Matrix:
[[7653   28]
 [  24   78]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```





Predicted islands after merging (count = 9). Example:

	Phage_ID	pred_island_start	pred_island_end	length
0	TemPhD_cluster_46242	2000	6500	4500
1	TemPhD_cluster_56564	49000	53000	4000
2	TemPhD_cluster_46209	47000	56500	9500
3	TemPhD_cluster_56212	32000	41000	9000
4	TemPhD_cluster_57210	48500	50500	2000

PIPELINE COMPLETE. Returning results (no CSVs written).

In [14]:

```
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
```

```
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=2500, step=625, k=4,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
```

```
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
```

```
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
```

```
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
```

```
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)
```

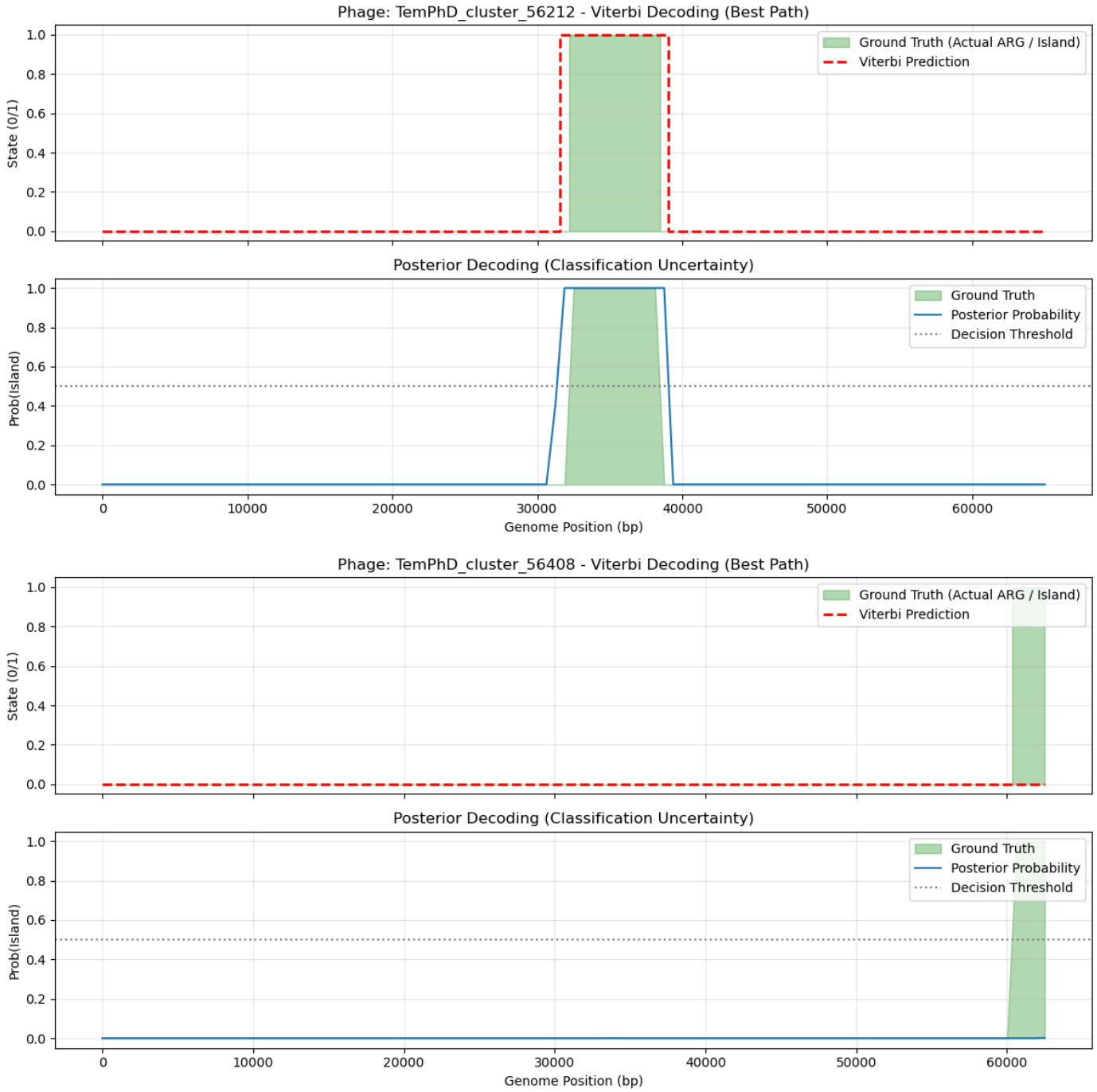
```
    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
```

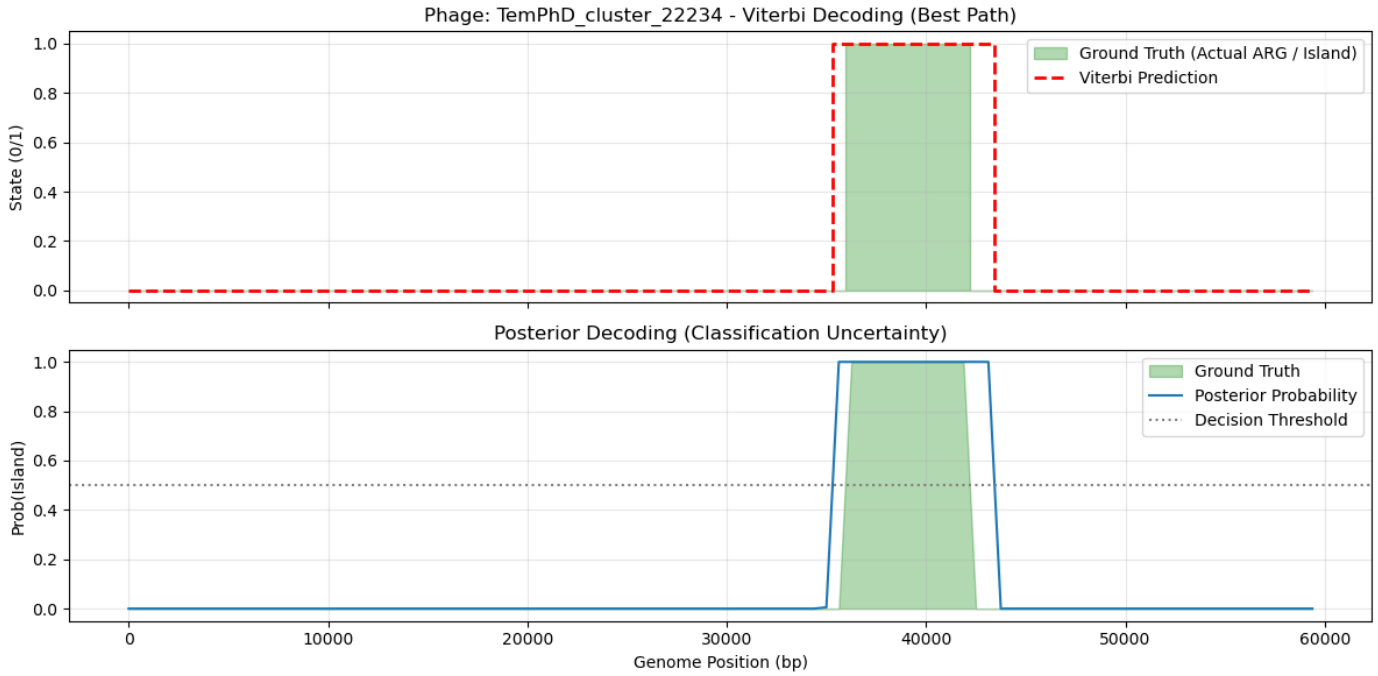
```
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
```

```
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=2500,
    step=625,
    k=4,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 75
Examples (first 10):
```

```
- TemPhD_cluster_56174: [{'start': 46875, 'end': 55000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03531067078074962}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_56907: [{'start': 45000, 'end': 53125, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.06216314412184076}] | #args=1 mobile_hits=5 genome_GC=
0.484
- TemPhD_cluster_56171: [{'start': 46875, 'end': 55000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03628955681571466}] | #args=1 mobile_hits=4 genome_GC=
0.304
- TemPhD_cluster_46242: [{'start': 625, 'end': 8750, 'length': 8125, 'arg_count': 1, 'mo
bile_count': 3, 'gc_shift': 0.03170652354739417}] | #args=2 mobile_hits=4 genome_GC=0.39
4
- TemPhD_cluster_25241: [{'start': 46875, 'end': 55000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.035600022501997064}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_34918: [{'start': 6875, 'end': 15000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.0701265220590987}] | #args=1 mobile_hits=4 genome_GC=0.
492
- TemPhD_cluster_39211: [{'start': 36250, 'end': 44375, 'length': 8125, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.009294934658649723}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_23154: [{'start': 46875, 'end': 55000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.03534436547629494}] | #args=1 mobile_hits=5 genome_GC=
0.304
- TemPhD_cluster_22243: [{'start': 36250, 'end': 44375, 'length': 8125, 'arg_count': 1,
'mobile_count': 3, 'gc_shift': 0.009220701326650982}] | #args=1 mobile_hits=3 genome_GC=
0.367
- TemPhD_cluster_23148: [{'start': 46875, 'end': 55000, 'length': 8125, 'arg_count': 1,
'mobile_count': 2, 'gc_shift': 0.035358989207686264}] | #args=1 mobile_hits=5 genome_GC=
0.304

[CHECKPOINT] LASSO selected 187 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.8214
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      0.99      0.99      6090
      Island       0.64      0.73      0.68        92

    accuracy                           0.99      6182
   macro avg       0.82      0.86      0.84      6182
weighted avg       0.99      0.99      0.99      6182

Confusion Matrix:
[[6053   37]
 [  25   67]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```






```
Predicted islands after merging (count = 16). Example:
              Phage_ID  pred_island_start  pred_island_end  length
0  TemPhD_cluster_56907              46875            50625    3750
1  TemPhD_cluster_46242               1250             6875    5625
2  TemPhD_cluster_56564              49375            53125    3750
3  TemPhD_cluster_26985               9375            12500    3125
4  TemPhD_cluster_59073              10000            13125    3125

PIPELINE COMPLETE. Returning results (no CSVs written).
```

In [15]:

```python
import numpy as np
import pandas as pd
from collections import Counter
import itertools
import warnings
import matplotlib.pyplot as plt
import random
import re
import os

# Sklearn & HMM Imports
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from hmmlearn import hmm

warnings.filterwarnings('ignore')

# ==========================================
# 1. HELPER FUNCTIONS
# ==========================================
def reverse_complement(seq):
    if seq is None:
        return None
    seq = str(seq)
```

```
    comp = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(comp)[::-1]

def get_kmers(sequence, k=4):
    if not sequence:
        return {}
    sequence = sequence.upper().replace('N', '')
    if len(sequence) < k:
        return {}
    bases = ['A', 'C', 'G', 'T']
    all_kmers = [''.join(p) for p in itertools.product(bases, repeat=k)]
    counts = Counter([sequence[i:i+k] for i in range(len(sequence)-k+1)])
    total = sum(counts.values())
    if total == 0:
        return {kmer: 0 for kmer in all_kmers}
    return {kmer: counts.get(kmer, 0)/total for kmer in all_kmers}

def get_gc(sequence):
    if not sequence:
        return 0.0
    s = sequence.upper()
    return (s.count('G') + s.count('C')) / len(s)

def is_mobile_element_annotation(annot_str, keywords=None):
    """
    Conservative mobile-element annotation checker using word-boundary regex.
    """
    if not isinstance(annot_str, str) or annot_str.strip() == "":
        return False
    if keywords is None:
        keywords = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]
    s = annot_str.lower()
    for kw in keywords:
        if re.search(r'\b' + re.escape(kw.lower()) + r'\b', s):
            return True
    return False

def visualize_decoding(phage_id, windows, y_true, y_pred_viterbi, y_prob_posterior):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
    ax1.fill_between(windows, 0, y_true, color='green', alpha=0.3, step='mid', label='Gr
    ax1.step(windows, y_pred_viterbi, where='mid', color='red', linestyle='--', linewidt
    ax1.set_ylabel("State (0/1)")
    ax1.set_title(f"Phage: {phage_id} - Viterbi Decoding (Best Path)")
    ax1.legend(loc='upper right')
    ax1.grid(True, alpha=0.3)

    ax2.fill_between(windows, 0, y_true, color='green', alpha=0.3, label='Ground Truth')
    ax2.plot(windows, y_prob_posterior, linewidth=1.5, label='Posterior Probability')
    ax2.axhline(y=0.5, color='gray', linestyle=':', label='Decision Threshold')
    ax2.set_ylabel("Prob(Island)")
    ax2.set_xlabel("Genome Position (bp)")
    ax2.set_title("Posterior Decoding (Classification Uncertainty)")
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

def merge_windows_to_ranges(positions, window_size, step):
    """Merge contiguous/touching window start positions into ranges (start, end)."""
    if len(positions) == 0:
        return []
    pos = np.sort(np.array(positions))
    ranges = []
    start = pos[0]
    prev = pos[0]
    for p in pos[1:]:
        if p <= prev + step:
            prev = p
            continue
        else:
            ranges.append((int(start), int(prev + window_size)))
            start = p
            prev = p
    ranges.append((int(start), int(prev + window_size)))
    return ranges

# ========================================
# 2. MAIN PROGRAM (strict island definition + HMM)
# ========================================
def run_pipeline(
    folder_path, db_name="TemPhD",
    window_size=900, step=300, k=3,
    island_radius=3000,            # +/- bp to group ARG-near windows
    min_args_for_island=1,         # require >= this many ARGs in candidate region (we k
    min_mobile_count=2,            # require >= this many mobile annotations in region O
    gc_shift_threshold=0.1,       # absolute GC difference threshold
    min_island_length_bp=2000,     # minimum merged island length
    mobile_kw_list=None,
    verbose=True
):
    """
    Detect ARG-enriched genomic islands: merged contiguous regions with >=1 ARG AND (mob
    Returns dictionary of results (no CSV writes).
    """
    if mobile_kw_list is None:
        mobile_kw_list = [
            'integrase', 'transposase', 'recombinase', 'resolvase',
            'insertion sequence', 'insertion-sequence', 'transposon',
            'integron', 'cassette'
        ]

    if verbose:
        print("--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-

    if 'DataFormat' not in globals():
        print("CRITICAL ERROR: 'DataFormat' class not found. Please run Part 1 code firs
        return

    data = DataFormat(folder_path, db_name)

    try:
        amr_df = data.__load_tsv__(data.amr_tsv)
        meta_df = data.__load_tsv__(data.meta_tsv)
```

```
        prot_df = data.__load_tsv__(data.protein_tsv)
    except Exception as e:
        print(f"Error loading data: {e}")
        return

    # normalize strand
    if 'Strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['Strand'].astype(str).str.strip()
    elif 'strand' in prot_df.columns:
        prot_df['Strand'] = prot_df['strand'].astype(str).str.strip()
    else:
        prot_df['Strand'] = '+'
        if verbose:
            print("WARNING: No 'Strand' column found — assuming '+' for all proteins.")

    # prepare phage lists
    hq_phages = meta_df[meta_df['Completeness'].isin(['High-quality', 'Complete'])]['Pha
    arg_phages = amr_df['Phage_id'].unique()
    valid_phages = list(set(hq_phages) & set(arg_phages))

    if len(valid_phages) == 0:
        print("No valid phages found after filtering completeness & ARG presence.")
        return

    train_phages, test_phages = train_test_split(valid_phages, test_size=0.20, random_st

    if verbose:
        print(f"\n[CHECKPOINT 1] DATASET PREPARATION")
        print(f"Total Valid Phages: {len(valid_phages)}")
        print(f"Training Set: {len(train_phages)} | Test Set: {len(test_phages)}")
        print("-------------------------------------------------")

    # collect global features + island summaries
    features_list = []
    island_summary = []

    if verbose:
        print(f"Extracting per-window features from {len(valid_phages)} phages...")

    for idx, phage_id in enumerate(valid_phages):
        df_seq = data.__load_fasta__(phage_id)
        if df_seq is None or df_seq.empty:
            continue
        seq = df_seq.iloc[0]['sequence']
        seq_len = len(seq)
        genome_gc = get_gc(seq)

        my_args = amr_df[amr_df['Phage_id'] == phage_id].copy()
        my_prots = prot_df[prot_df['Phage_ID'] == phage_id].copy()

        # arg protein ids
        if 'Protein_id' in my_args.columns:
            arg_prot_ids = set(my_args['Protein_id'].astype(str))
        else:
            arg_prot_ids = set()

        # build coordinates for ARG-associated proteins (if present)
        arg_coords = []
        for _, r in my_prots[my_prots['Protein_ID'].astype(str).isin(arg_prot_ids)].iter
```

```
            try:
                s0 = max(0, int(r['Start']) - 1)
                e0 = min(seq_len, int(r['Stop']))
                arg_coords.append((s0, e0))
            except:
                continue
        arg_midpoints = [(s+e)//2 for (s,e) in arg_coords]

        # compute mobile positions from annotations
        mobile_positions = []
        for _, prow in my_prots.iterrows():
            annot_fields = []
            for col in ['Product','Annotation','Function','Description','product','annot
                if col in my_prots.columns and pd.notna(prow.get(col, None)):
                    annot_fields.append(str(prow.get(col,'')))
            annot_concat = " ".join(annot_fields).strip()
            if is_mobile_element_annotation(annot_concat, mobile_kw_list):
                try:
                    ps0 = max(0, int(prow['Start']) - 1)
                    pe0 = min(seq_len, int(prow['Stop']))
                    mobile_positions.append((ps0, pe0))
                except:
                    continue

        # per-phage per-window temporary store
        phage_rows = []

        # sliding windows: collect per-window stats but do NOT finalize label yet
        for i in range(0, max(1, seq_len - window_size + 1), step):
            w_seq = seq[i:i+window_size]
            w_start, w_end = i, min(i + window_size, seq_len)

            # ARG overlaps in window (count)
            arg_overlap_count = 0
            for (a0,a1) in arg_coords:
                if a0 < w_end and a1 > w_start:
                    arg_overlap_count += 1

            # mobile overlaps in window (count)
            mobile_overlap_count = 0
            for (m0,m1) in mobile_positions:
                if m0 < w_end and m1 > w_start:
                    mobile_overlap_count += 1

            # near_any_arg: center within island_radius of any ARG midpoint
            center = (w_start + w_end)//2
            near_any_arg = any(abs(mp - center) <= island_radius for mp in arg_midpoints

            row = {
                'Phage_ID': phage_id,
                'Window_Start': w_start,
                'Window_End': w_end,
                'Local_GC': get_gc(w_seq),
                'kmer': None,  # placeholder, we'll expand k-mers later if needed
                'arg_overlap_count': arg_overlap_count,
                'mobile_overlap_count': mobile_overlap_count,
                'near_any_arg': int(near_any_arg),
                # label to be set after merged-range evaluation
                'Label': 0
```

```
            }
            phage_rows.append(row)

        # convert to df for this phage and find candidate windows near any ARG
        phage_df = pd.DataFrame(phage_rows)
        candidate_positions = phage_df[phage_df['near_any_arg'] == 1]['Window_Start'].va
        merged_candidate_ranges = merge_windows_to_ranges(candidate_positions, window_si

        # Evaluate each merged candidate range and accept only those that satisfy final
        #   (1) region contains >= min_args_for_island ARGs
        #   (2) region length >= min_island_length_bp
        #   (3) and (mobile_count >= min_mobile_count OR GC shift >= gc_shift_threshold)
        accepted_ranges = []
        for (rs, re) in merged_candidate_ranges:
            # count ARGs whose midpoints fall in [rs, re)
            arg_count_in_region = sum(1 for mp in arg_midpoints if (mp >= rs and mp < re
            # count mobile hits overlapping region
            mobile_count_in_region = sum(1 for (m0,m1) in mobile_positions if not (m1 <=
            # compute GC shift for the full region (extract sequence safely)
            try:
                region_seq = seq[rs:re]
                region_gc = get_gc(region_seq)
            except:
                region_gc = genome_gc
            gc_shift = abs(region_gc - genome_gc)
            region_len = re - rs

            # acceptance criteria
            meets_arg = arg_count_in_region >= max(1, min_args_for_island)  # ensure at
            meets_mobile_or_gc = (mobile_count_in_region >= min_mobile_count) or (gc_shi
            meets_length = region_len >= min_island_length_bp

            if meets_arg and meets_mobile_or_gc and meets_length:
                accepted_ranges.append({
                    'start': rs,
                    'end': re,
                    'length': region_len,
                    'arg_count': arg_count_in_region,
                    'mobile_count': mobile_count_in_region,
                    'gc_shift': gc_shift
                })

        # set labels for windows that fall inside accepted ranges
        if len(accepted_ranges) > 0:
            for ar in accepted_ranges:
                mask = (phage_df['Window_Start'] >= ar['start']) & (phage_df['Window_End
                phage_df.loc[mask, 'Label'] = 1

        # record island summary info
        island_summary.append({
            'Phage_ID': phage_id,
            'num_accepted_islands': len(accepted_ranges),
            'accepted_islands': accepted_ranges,
            'num_args': len(arg_midpoints),
            'num_mobile_positions': len(mobile_positions),
            'genome_gc': genome_gc
        })

        # expand k-mers and other features now and append to global list
```

```
        for _, prow in phage_df.iterrows():
            wstart = int(prow['Window_Start'])
            wend = int(prow['Window_End'])
            wseq = seq[wstart:wend]
            row_features = {
                'Phage_ID': phage_id,
                'Window_Start': wstart,
                'Label': int(prow['Label']),
                'Local_GC': prow['Local_GC'],
                'Arg_Count_Window': prow['arg_overlap_count'],
                'Mobile_Count_Window': prow['mobile_overlap_count'],
                'Protein_Density': 0  # we'll fill protein density below
            }
            # add kmer features (k up to k param)
            kmer_feats = get_kmers(wseq, k=k)
            row_features.update(kmer_feats)

            # compute overlapped proteins to approximate protein density and protein sta
            overlaps = []
            for _, p in my_prots.iterrows():
                try:
                    ps0 = max(0, int(p['Start']) - 1)
                    pe0 = min(seq_len, int(p['Stop']))
                except:
                    continue
                if ps0 < wend and pe0 > wstart:
                    overlaps.append(p)
            overlapped_prots = pd.DataFrame(overlaps) if len(overlaps) else pd.DataFrame
            orf_seqs = []
            for _, op in overlapped_prots.iterrows():
                try:
                    ps0 = max(0, int(op['Start']) - 1)
                    pe0 = min(seq_len, int(op['Stop']))
                except:
                    continue
                subseq = seq[ps0:pe0]
                if str(op.get('Strand', '+')).strip() in ['-', '-1']:
                    subseq = reverse_complement(subseq)
                orf_seqs.append(subseq)
            row_features['Protein_Density'] = len(orf_seqs)

            # simple protein property means if present
            for c in ['Aromaticity', 'Instability_index', 'Isoelectric_point']:
                if c in overlapped_prots.columns and len(overlapped_prots) > 0:
                    vals = pd.to_numeric(overlapped_prots[c], errors='coerce').dropna()
                    row_features[c] = vals.mean() if len(vals) > 0 else 0.0
                else:
                    row_features[c] = 0.0

            features_list.append(row_features)

    # create features dataframe
    df = pd.DataFrame(features_list)
    if df.empty:
        print("No features extracted. Exiting.")
        return

    # create island summary dataframe for diagnostics
    island_df = pd.DataFrame(island_summary)
```

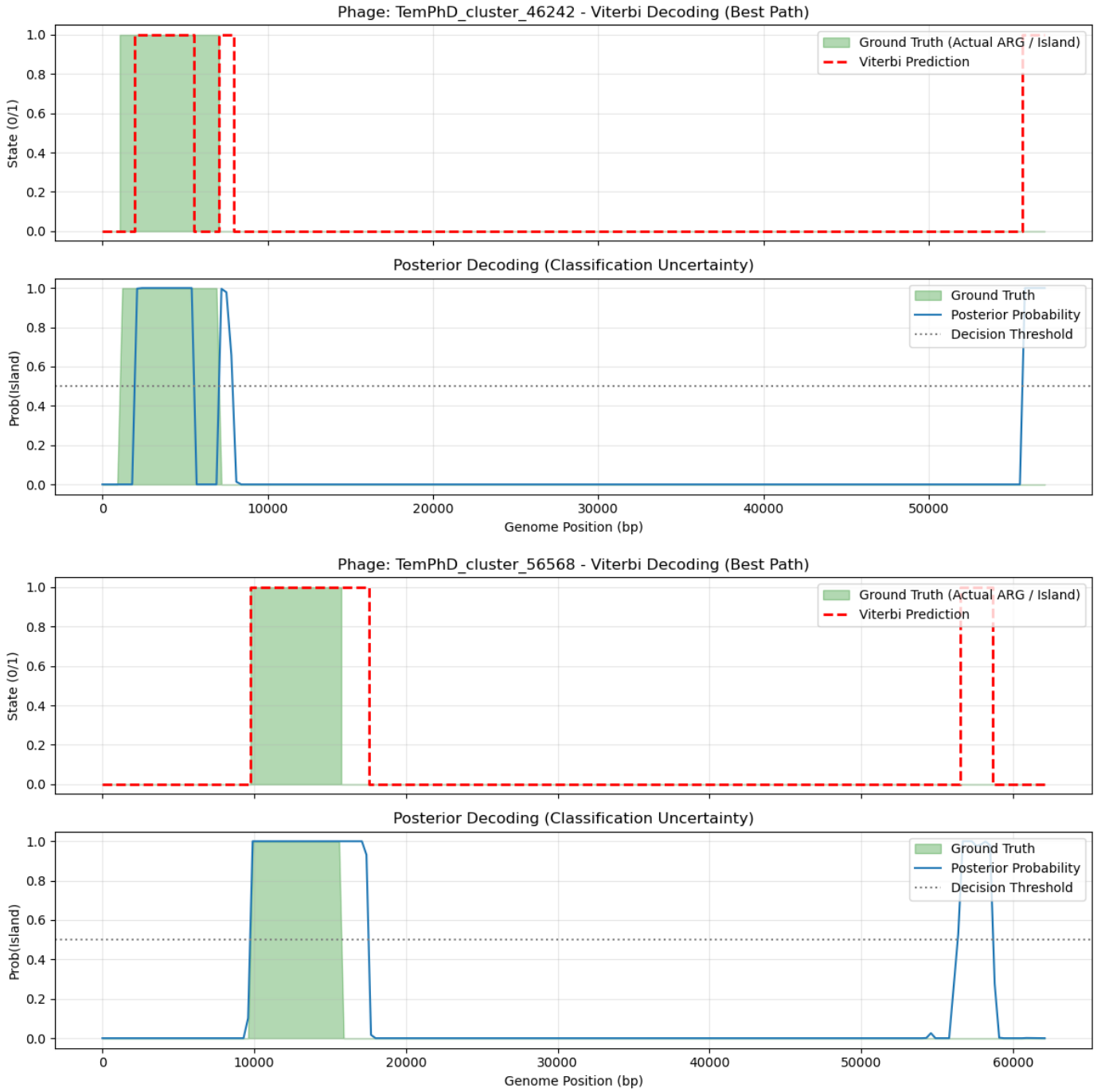
```
    # Diagnostic printout
    if verbose:
        total_with_islands = island_df[island_df['num_accepted_islands'] > 0].shape[0]
        print("\n=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===")
        print(f"Phages analyzed: {len(island_df)}. Phages with ≥1 accepted island: {tota
        if total_with_islands > 0:
            print("Examples (first 10):")
            sample_rows = island_df[island_df['num_accepted_islands'] > 0].head(10)
            for _, r in sample_rows.iterrows():
                print(f"- {r['Phage_ID']}: {r['accepted_islands']} | #args={r['num_args'

    # Now split into train/test by phage using earlier train_phages/test_phages
    df_train = df[df['Phage_ID'].isin(train_phages)]
    df_test = df[df['Phage_ID'].isin(test_phages)]

    # Preprocess: drop id columns and impute/scale
    drop_cols = ['Phage_ID', 'Label', 'Window_Start']
    imputer = SimpleImputer(strategy='mean')
    scaler = StandardScaler()

    Xtrain_raw = df_train.drop(columns=drop_cols)
    Xtest_raw = df_test.drop(columns=drop_cols)

    Xtrain_imp = imputer.fit_transform(Xtrain_raw)
    Xtest_imp  = imputer.transform(Xtest_raw)

    Xtrain_scaled = scaler.fit_transform(Xtrain_imp)
    Xtest_scaled  = scaler.transform(Xtest_imp)

    # LASSO for feature selection
    lasso = LassoCV(cv=5, random_state=42, max_iter=5000)
    lasso.fit(Xtrain_scaled, df_train['Label'].values)

    coef_mask = np.abs(lasso.coef_) > 1e-8
    if coef_mask.sum() < 5:
        idxs = np.argsort(-np.abs(lasso.coef_))[:5]
        coef_mask = np.zeros_like(lasso.coef_, dtype=bool)
        coef_mask[idxs] = True

    selected_indices = np.where(coef_mask)[0]
    X_train = Xtrain_scaled[:, selected_indices]
    X_test  = Xtest_scaled[:, selected_indices]

    if verbose:
        print(f"\n[CHECKPOINT] LASSO selected {len(selected_indices)} features.")

    # PCA
    n_components = min(50, X_train.shape[1])
    if n_components < 1:
        print("Not enough features selected for PCA. Exiting.")
        return
    pca = PCA(n_components=n_components)
    X_train = pca.fit_transform(X_train)
    y_train = df_train['Label'].values
    X_test = pca.transform(X_test)
    y_test = df_test['Label'].values

    if verbose:
```

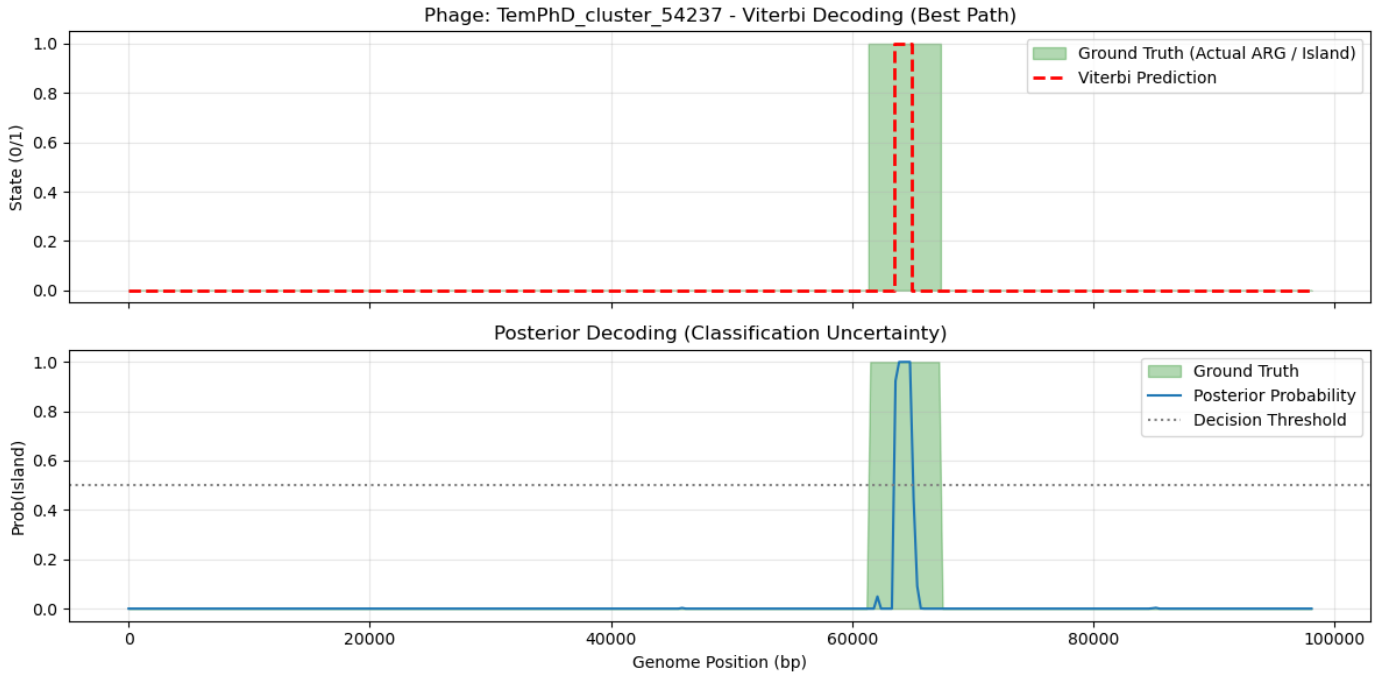
```
        print(f"\n[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE")
        print(f"PCA Variance Explained: {np.sum(pca.explained_variance_ratio_):.4f}")
        print("-------------------------------------------------")

    # HMM training
    X_bg = X_train[y_train == 0]
    X_island = X_train[y_train == 1]

    if len(X_island) < 5:
        print("CRITICAL WARNING: not enough island examples in training; aborting.")
        return

    model = hmm.GaussianHMM(n_components=2, covariance_type="full", init_params="", verb
    model.startprob_ = np.array([0.99, 0.01])
    model.transmat_ = np.array([[0.98, 0.02], [0.05, 0.95]])
    model.means_ = np.array([X_bg.mean(axis=0), X_island.mean(axis=0)])
    cov_bg = np.cov(X_bg.T) + np.eye(model.means_.shape[1]) * 0.1
    cov_island = np.cov(X_island.T) + np.eye(model.means_.shape[1]) * 0.1
    model.covars_ = np.array([cov_bg, cov_island])
    model.n_features = model.means_.shape[1]

    if verbose:
        print(f"\n[CHECKPOINT 3] MODEL TRAINED")

    # predict on test
    lengths_test = df_test.groupby('Phage_ID').size().values
    if X_test.shape[0] == 0:
        print("No test data available after split.")
        return

    y_pred_viterbi = model.predict(X_test, lengths_test)

    if verbose:
        print("\n=== EVALUATION ON TEST DATASET ===")
        print(classification_report(y_test, y_pred_viterbi, target_names=['Background',
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred_viterbi))
        print("\n[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)")

    # Visualize some phages
    test_phages_with_island = df_test[df_test['Label'] == 1]['Phage_ID'].unique()
    if len(test_phages_with_island) > 0 and verbose:
        viz_phages = random.sample(list(test_phages_with_island), min(3, len(test_phages
        for pid in viz_phages:
            mask = (df_test['Phage_ID'] == pid).values
            X_p = X_test[mask]
            windows = df_test[mask]['Window_Start'].values
            y_true_p = df_test[mask]['Label'].values
            y_vit = model.predict(X_p, [len(X_p)])
            try:
                y_post = model.predict_proba(X_p, [len(X_p)])[:, 1]
            except:
                y_post = np.zeros(len(X_p))
            visualize_decoding(pid, windows, y_true_p, y_vit, y_post)
    else:
        if verbose:
            print("No ARG-enriched islands found in Test Set to visualize.")

    # merge predicted windows into islands (post-HMM) and filter by min length
```

```
    df_test = df_test.copy().reset_index(drop=True)
    df_test['Predicted'] = y_pred_viterbi.astype(int)
    all_pred_islands = []
    for pid in df_test['Phage_ID'].unique():
        df_p = df_test[df_test['Phage_ID'] == pid]
        pred_positions = df_p[df_p['Predicted'] == 1]['Window_Start'].values
        merged = merge_windows_to_ranges(pred_positions, window_size=window_size, step=s
        merged_filtered = [r for r in merged if (r[1] - r[0]) >= min_island_length_bp]
        for (s,e) in merged_filtered:
            all_pred_islands.append({'Phage_ID': pid, 'pred_island_start': s, 'pred_isla
    pred_islands_df = pd.DataFrame(all_pred_islands)

    if verbose:
        if pred_islands_df.empty:
            print("\nNo predicted islands after merging + length filtering.")
        else:
            print(f"\nPredicted islands after merging (count = {len(pred_islands_df)}).
            print(pred_islands_df.head())

    if verbose:
        print("\nPIPELINE COMPLETE. Returning results (no CSVs written).")

    return {
        'features_df': df,
        'accepted_island_summary': island_df,
        'predicted_islands': pred_islands_df,
        'hmm_model': model,
        'pca': pca,
        'lasso': lasso,
        'scaler': scaler,
        'imputer': imputer
    }

# Example run (adjust path)
my_folder = r"C:\Users\subak\OneDrive\Documents\Comp Bio"
outputs = run_pipeline(
    my_folder,
    window_size=900,
    step=300,
    k=3,
    island_radius=3000,
    min_args_for_island=1,
    min_mobile_count=2,
    gc_shift_threshold=0.1,
    min_island_length_bp=2000,
    verbose=True
)
```

```
--- STARTING STRICT ARG-ENRICHED ISLAND DETECTION + HMM PIPELINE (strand-aware) ---

[CHECKPOINT 1] DATASET PREPARATION
Total Valid Phages: 328
Training Set: 262 | Test Set: 66
--------------------------------------------------
Extracting per-window features from 328 phages...

=== DIAGNOSTIC: ACCEPTED ISLANDS (post-merge + criteria) ===
Phages analyzed: 328. Phages with ≥1 accepted island: 31
Examples (first 10):
```

```
- TemPhD_cluster_46242: [{'start': 1200, 'end': 7800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.03411118555205617}] | #args=2 mobile_hits=4 genome_GC=0.394
- TemPhD_cluster_39211: [{'start': 37200, 'end': 43800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.0026795500432650887}] | #args=1 mobile_hits=3 genome_GC=0.367
- TemPhD_cluster_22243: [{'start': 37200, 'end': 43800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.002605316711266348}] | #args=1 mobile_hits=3 genome_GC=0.367
- TemPhD_cluster_22244: [{'start': 37200, 'end': 43800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.002701459772140924}] | #args=1 mobile_hits=3 genome_GC=0.367
- TemPhD_cluster_46209: [{'start': 48000, 'end': 54600, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.016182230890131155}] | #args=1 mobile_hits=5 genome_GC=0.354
- TemPhD_cluster_22237: [{'start': 37200, 'end': 43800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.00263736439822454}] | #args=1 mobile_hits=3 genome_GC=0.367
- TemPhD_cluster_54409: [{'start': 27300, 'end': 33900, 'length': 6600, 'arg_count': 1, 'mobile_count': 3, 'gc_shift': 0.0013731169499384133}] | #args=1 mobile_hits=4 genome_GC=0.479
- TemPhD_cluster_54261: [{'start': 37800, 'end': 45000, 'length': 7200, 'arg_count': 2, 'mobile_count': 2, 'gc_shift': 0.0554286975736033}] | #args=3 mobile_hits=6 genome_GC=0.359
- TemPhD_cluster_55771: [{'start': 61500, 'end': 70500, 'length': 9000, 'arg_count': 2, 'mobile_count': 3, 'gc_shift': 0.04482858753710073}] | #args=2 mobile_hits=7 genome_GC=0.398
- TemPhD_cluster_22235: [{'start': 37200, 'end': 43800, 'length': 6600, 'arg_count': 1, 'mobile_count': 2, 'gc_shift': 0.002685435928661828}] | #args=1 mobile_hits=3 genome_GC=0.367

[CHECKPOINT] LASSO selected 52 features.

[CHECKPOINT 2] DIMENSIONALITY REDUCTION COMPLETE
PCA Variance Explained: 0.9998
--------------------------------------------------

[CHECKPOINT 3] MODEL TRAINED

=== EVALUATION ON TEST DATASET ===
              precision    recall  f1-score   support

  Background       1.00      0.99      0.99     13073
      Island       0.35      0.81      0.49       120

    accuracy                           0.98     13193
   macro avg       0.67      0.90      0.74     13193
weighted avg       0.99      0.98      0.99     13193

Confusion Matrix:
[[12894   179]
 [   23    97]]

[CHECKPOINT 4] VISUALIZING POSTERIOR DECODING (Uncertainty Analysis)
```





Predicted islands after merging (count = 15). Example:

	Phage_ID	pred_island_start	pred_island_end	length
0	TemPhD_cluster_46242	2100	6300	4200
1	TemPhD_cluster_46242	55800	57900	2100
2	TemPhD_cluster_56564	49200	52500	3300
3	TemPhD_cluster_46209	6000	8100	2100
4	TemPhD_cluster_46209	48000	56400	8400

PIPELINE COMPLETE. Returning results (no CSVs written).

In [ ]: