"BioQuery Local" - Self-Contained Natural Language Bioinformatics Tool to do EMBOSS Analyses

This is a suggested AI-driven alternative to Final Group Project. Feel free to do this as described, add, change, or improve. Email me if you decide to select this alternative to the Web Analysis Tool project. You can work on your own, or in groups of two for this project.

There will be all the same "Demo" requirements and timeline as defined in main project, but also requires (1) puting your finished work in GitHub, and (2) anyone else in the class (including me!) should be able to download and run what you present.

Initial Setup (Pre-Week 1)

```shell
# Create conda environment
conda create -n bioquery python=3.10
conda activate bioquery

# Install bioinformatics tools
conda install -c bioconda emboss biopython
conda install -c conda-forge streamlit pandas

# Install Ollama (macOS)
brew install ollama

# Pull a lightweight model (Phi-3 is small and fast)
ollama pull phi3:mini  # 2GB model, good for bioinformatics
# Alternative: ollama pull llama3.2:3b  # Slightly larger but more capable

# Install Python Ollama client
pip install ollama
```

Week 1: EMBOSS Integration & Basic Framework

Create a wrapper for EMBOSS tools with natural language mapping:

Python

```python
# emboss_wrapper.py
import subprocess
import tempfile
import os
from pathlib import Path

class EMBOSSWrapper:
    """Wrapper for common EMBOSS tools"""

    def __init__(self):
        # Map natural language to EMBOSS commands
        self.tool_map = {
            'translate': 'transeq',
            'reverse': 'revseq',
            'orf': 'getorf',
            'align': 'needle',
            'pattern': 'fuzznuc',
            'restriction': 'restrict',
            'shuffle': 'shuffleseq',
            'info': 'infoseq',
            'sixframe': 'sixpack'
        }

        # Verify EMBOSS installation
        self.check_emboss()

    def check_emboss(self):
        """Verify EMBOSS tools are available"""
        try:
            result = subprocess.run(['embossversion'],
                                    capture_output=True, text=True)
            print(f"EMBOSS found: {result.stdout.strip()}")
            return True
        except FileNotFoundError:
            print("EMBOSS not found. Please install with: conda install -c
bioconda emboss")
            return False
    def run_emboss_tool(self, tool, input_seq, **kwargs):
        """Generic EMBOSS tool runner"""
        with tempfile.NamedTemporaryFile(mode='w', suffix='.fasta', 
delete=False) as f_in:
            # Write input sequence
            f_in.write(f">Query\n{input_seq}\n")
            f_in.flush()

            with tempfile.NamedTemporaryFile(mode='r', suffix='.out', 
delete=False) as f_out:
                cmd = [tool, f_in.name, f_out.name]

                # Add any additional parameters
                for key, value in kwargs.items():
                    cmd.extend([f'-{key}', str(value)])

                try:
                    result = subprocess.run(cmd, capture_output=True, 
text=True)

                    # Read output
                    with open(f_out.name, 'r') as output:
                        return output.read()

                except Exception as e:
                    return f"Error running {tool}: {e}"
                finally:
                    # Cleanup temp files
                    os.unlink(f_in.name)
                    os.unlink(f_out.name)

    def translate(self, sequence, frame=1):
        """Translate DNA to protein using transeq"""
        return self.run_emboss_tool('transeq', sequence, frame=frame)

    def reverse_complement(self, sequence):
        """Get reverse complement using revseq"""
        return self.run_emboss_tool('revseq', sequence)
    def find_orfs(self, sequence, minsize=100):
        """Find ORFs using getorf"""
        return self.run_emboss_tool('getorf', sequence, minsize=minsize)

    def find_pattern(self, sequence, pattern):
        """Find sequence pattern using fuzznuc"""
        return self.run_emboss_tool('fuzznuc', sequence, pattern=pattern)

    def restriction_sites(self, sequence):
        """Find restriction sites using restrict"""
        return self.run_emboss_tool('restrict', sequence, enzymes='all')
```

Week 2: Ollama LLM Integration

Create a local LLM query parser:

Python
```python
# llm_parser.py
import ollama
import json
import re

class LocalLLMParser:
    """Parse natural language queries using Ollama"""

    def __init__(self, model='phi3:mini'):
        self.model = model
        self.client = ollama.Client()

        # Define available tools and their descriptions
        self.tools_description = """
        Available bioinformatics tools:
        - translate: Convert DNA to protein sequence
        - reverse: Get reverse complement of DNA
        - find_orfs: Find open reading frames in DNA
        - align: Align two sequences
        - pattern: Find specific pattern in sequence
        - restriction: Find restriction enzyme sites
        - gc_content: Calculate GC content percentage
        - sixframe: Translate in all six reading frames
        """

    def parse_query(self, user_query):
        """Use LLM to parse natural language query"""

        prompt = f"""You are a bioinformatics assistant. Parse this query
and extract:
1. The tool to use (from the list below)
2. The sequence(s) or gene name(s) involved
3. Any additional parameters

{self.tools_description}

User query: {user_query}

Respond in JSON format:
{{
    "tool": "tool_name",
    "sequence": "DNA_or_protein_sequence",
    "gene_name": "gene_if_mentioned",
    "parameters": {{}}
}}

If the query contains a DNA sequence (letters ATCG), extract it.
Be concise and accurate."""

        try:
            response = self.client.generate(
                model=self.model,
                prompt=prompt,
                stream=False
            )

            # Extract JSON from response
            response_text = response['response']

            # Try to parse JSON
            json_match = re.search(r'\{.*\}', response_text, re.DOTALL)
            if json_match:
                return json.loads(json_match.group())
            else:
                # Fallback to simple parsing
                return self.simple_parse(user_query)

        except Exception as e:
            print(f"LLM parsing failed: {e}")
            return self.simple_parse(user_query)

    def simple_parse(self, query):
        """Fallback simple parser if LLM fails"""
        query_lower = query.lower()

        # Extract DNA sequences
        seq_pattern = r'[ATCG]{10,}'
        seq_match = re.search(seq_pattern, query.upper())

        result = {
            "tool": None,
            "sequence": seq_match.group() if seq_match else None,
            "gene_name": None,
            "parameters": {}
        }

        # Determine tool based on keywords
        if 'translate' in query_lower:
            result['tool'] = 'translate'
        elif 'reverse' in query_lower or 'complement' in query_lower:
            result['tool'] = 'reverse'
        elif 'orf' in query_lower or 'reading frame' in query_lower:
            result['tool'] = 'find_orfs'
        elif 'pattern' in query_lower or 'find' in query_lower:
            result['tool'] = 'pattern'
        elif 'restriction' in query_lower or 'enzyme' in query_lower:
            result['tool'] = 'restriction'
        elif 'gc' in query_lower:
            result['tool'] = 'gc_content'

        return result
```

Week 3: BioPython Integration for Additional Features

Add BioPython functionality for things EMBOSS doesn't cover easily:

```python
# bio_tools.py
from Bio import Entrez, SeqIO
from Bio.Seq import Seq
from Bio.SeqUtils import GC
import os

class BioTools:
    """Additional tools using BioPython"""

    def __init__(self):
        Entrez.email = "student@university.edu"

    def fetch_sequence(self, gene_name, db='nucleotide'):
        """Fetch sequence from NCBI (if internet available)"""
        try:
            # Search for the gene
            handle = Entrez.esearch(db=db, term=gene_name, retmax=1)
            record = Entrez.read(handle)
            handle.close()

            if record["IdList"]:
                # Fetch the sequence
                handle = Entrez.efetch(db=db,
                                       id=record["IdList"][0],
                                       rettype="fasta",
                                  retmode="text")
                sequence = SeqIO.read(handle, "fasta")
                handle.close()
                return str(sequence.seq)
            else:
                return None
        except:
            return None

    def gc_content(self, sequence):
        """Calculate GC content"""
        seq = Seq(sequence.upper().replace('\n', ''))
        gc_percent = GC(seq)

        # Also calculate by position
        window_size = 10
        gc_windows = []
        for i in range(0, len(seq) - window_size):
            window = seq[i:i+window_size]
            gc_windows.append(GC(window))

        return {
            'overall_gc': gc_percent,
            'length': len(seq),
            'gc_windows': gc_windows,
            'min_gc': min(gc_windows) if gc_windows else 0,
            'max_gc': max(gc_windows) if gc_windows else 0
        }

    def sequence_stats(self, sequence):
        """Get basic sequence statistics"""
        seq = Seq(sequence.upper().replace('\n', ''))

        return {
            'length': len(seq),
            'a_count': seq.count('A'),
            't_count': seq.count('T'),
            'g_count': seq.count('G'),
            'c_count': seq.count('C'),
            'gc_content': GC(seq),
            'at_content': 100 - GC(seq)
        }
```

Week 4: Main Application Logic

Combine everything into a cohesive application:

```python
# bioquery_local.py
from emboss_wrapper import EMBOSSWrapper
from llm_parser import LocalLLMParser
from bio_tools import BioTools
import re

class BioQueryLocal:
    """Main application combining all components"""

    def __init__(self):
        self.emboss = EMBOSSWrapper()
        self.llm = LocalLLMParser()
        self.biotools = BioTools()

        # Example sequences for testing
        self.example_sequences = {
            'test_dna': 'ATGGCGAATTACGTAGCTAGCTAGCGCGCTATAGCGCGCTAA',
            'brca1_fragment': 'ATGGATTTATCTGCTCTTCGCGTTGAAGAAGTACAAAATGTCA',
            'p53_fragment': 'ATGGAGGAGCCGCAGTCAGATCCTAGCGTCGAGCCCCCTCTGA'
        }

    def process_query(self, query):
        """Process natural language query"""

        # Parse query with LLM
        parsed = self.llm.parse_query(query)
        # Get sequence (from query or from examples)
        sequence = parsed.get('sequence')

        if not sequence and parsed.get('gene_name'):
            # Try to fetch from NCBI
            sequence = self.biotools.fetch_sequence(parsed['gene_name'])

            # Fallback to examples
            if not sequence:
                gene_lower = parsed['gene_name'].lower()
                for key, seq in self.example_sequences.items():
                    if gene_lower in key.lower():
                        sequence = seq
                        break

        if not sequence:
            # Try to extract from query directly
            seq_match = re.search(r'[ATCG]{10,}', query.upper())
            if seq_match:
                sequence = seq_match.group()

        if not sequence:
            return {
                'success': False,
                'error': 'No sequence found. Please provide a DNA sequence
or gene name.',
                'parsed': parsed
            }

        # Execute the appropriate tool
        tool = parsed.get('tool')
        result = None

        if tool == 'translate':
            result = self.emboss.translate(sequence)
        elif tool == 'reverse':
            result = self.emboss.reverse_complement(sequence)
        elif tool == 'find_orfs':
            result = self.emboss.find_orfs(sequence)
        elif tool == 'pattern':
            pattern = parsed.get('parameters', {}).get('pattern', 'ATG')
            result = self.emboss.find_pattern(sequence, pattern)
        elif tool == 'restriction':
            result = self.emboss.restriction_sites(sequence)
        elif tool == 'gc_content':
            result = self.biotools.gc_content(sequence)
        else:
            result = f"Unknown tool: {tool}"

        return {
            'success': True,
            'tool': tool,
            'sequence': sequence[:50] + '...' if len(sequence) > 50 else
sequence,
            'result': result,
            'parsed': parsed
        }

    def get_examples(self):
        """Return example queries for users"""
        return [
            "Translate the sequence ATGGCGAATTACGTAGCT",
            "What is the reverse complement of ATCGATCGATCG?",
            "Find open reading frames in " +
self.example_sequences['test_dna'],
            "Calculate GC content of GCGCGCATATATATGCGCGC",
            "Find restriction sites in GAATTCGCGGCCGCTCTAGAACTAGTGGATC",
            "Find ATG patterns in ATGATGATGATGATGATG",
            "Translate BRCA1 fragment",
            "Get six-frame translation of p53_fragment"
        ]
```

Week 5: Streamlit Web Interface

Create a polished web interface:

Python
```
# app.py
import streamlit as st
from bioquery_local import BioQueryLocal
import json

st.set_page_config(page_title="BioQuery Local", page_icon="🧬",
layout="wide")

# Initialize session state
if 'bq' not in st.session_state:
    st.session_state.bq = BioQueryLocal()
if 'history' not in st.session_state:
    st.session_state.history = []

# Header
st.title("🧬 BioQuery Local")
st.markdown("Natural language bioinformatics - running entirely on your
machine!")

# Sidebar with information
with st.sidebar:
    st.markdown("### 🛠️ Available Tools")
    st.markdown("""
    - **Translate**: DNA to protein
    - **Reverse Complement**: Reverse and complement DNA
    - **Find ORFs**: Locate open reading frames
    - **GC Content**: Calculate GC percentage
    - **Restriction Sites**: Find enzyme cut sites
    - **Pattern Search**: Find sequence patterns
    """)

    st.markdown("### 📚 Example Sequences")
    if st.button("Load Test DNA"):
        st.session_state.query =
st.session_state.bq.example_sequences['test_dna']
    if st.button("Load BRCA1 Fragment"):
        st.session_state.query =
st.session_state.bq.example_sequences['brca1_fragment']
```

```
    if st.button("Load P53 Fragment"):
        st.session_state.query =
st.session_state.bq.example_sequences['p53_fragment']

    st.markdown("### 💡 Example Queries")
    examples = st.session_state.bq.get_examples()
    for ex in examples[:5]:
        if st.button(ex[:30] + "...", key=ex):
            st.session_state.query = ex

# Main interface
col1, col2 = st.columns([3, 1])

with col1:
    query = st.text_area("Enter your bioinformatics question or paste a
sequence:",
                        value=st.session_state.get('query', ''),
                        height=100,
                        placeholder="Try: 'Translate ATGGCGAAT' or 'Find
ORFs in this sequence...'")

with col2:
    st.markdown("<br>", unsafe_allow_html=True)
    process_button = st.button("🔬 Process Query", type="primary",
use_container_width=True)
    clear_button = st.button("🗑️ Clear", use_container_width=True)

if clear_button:
    st.session_state.query = ""
    st.session_state.history = []
    st.rerun()

if process_button and query:
    with st.spinner("🤔 Understanding your query with local LLM..."):
        result = st.session_state.bq.process_query(query)
        st.session_state.history.append((query, result))

    if result['success']:
        # Success message
        st.success(f"✅ Executed: {result['tool']}")

        # Show results
        tab1, tab2, tab3 = st.tabs(["📊 Results", "🔍 How it worked", "📝
Raw Output"])

        with tab1:
            st.markdown("### Results")

            # Format output based on tool type
            if result['tool'] == 'gc_content' and
isinstance(result['result'], dict):
                col1, col2, col3 = st.columns(3)
                with col1:
                    st.metric("Overall GC%",
f"{result['result']['overall_gc']:.2f}%")
                with col2:
                    st.metric("Sequence Length", result['result']['length'])
                with col3:
                    st.metric("Min/Max GC",
                             f"{result['result'].get('min_gc', 0):.1f}% /
{result['result'].get('max_gc', 0):.1f}%")
            else:
                # Display as code block for sequence results
                if isinstance(result['result'], str):
                    st.code(result['result'], language='text')
                else:
                    st.json(result['result'])

        with tab2:
            st.markdown("### Behind the Scenes")
            st.markdown(f"""
            1. **Query received**: "{query[:100]}..."
            2. **LLM parsing**: Local Ollama model identified the intent
            3. **Tool selected**: `{result['tool']}`
            4. **Sequence extracted**: {result['sequence']}
            5. **EMBOSS/BioPython executed**: Tool ran locally on your
machine
            6. **Results returned**: Formatted for display
            """)

            with st.expander("🧠 LLM Parse Details"):
                st.json(result['parsed'])

        with tab3:
            st.markdown("### Raw Output")
            st.text(str(result))
    else:
        st.error(f"❌ {result['error']}")
        with st.expander("Debug Information"):
            st.json(result['parsed'])

# History section
if st.session_state.history:
    st.markdown("---")
    st.markdown("### 📜 Query History")
    for i, (q, r) in enumerate(reversed(st.session_state.history[-5:])):
        with st.expander(f"Query {len(st.session_state.history)-i}:
{q[:50]}..."):
            st.json(r)

# Footer
st.markdown("---")
st.caption("🏫 BME 110 - Running locally with EMBOSS, BioPython, and
Ollama")
```

Testing Script for Students

Python
```
# test_bioquery.py
"""Test script to verify all components are working"""

def test_installation():
    """Test that all components are installed correctly"""
    print("Testing BioQuery Local Installation...\n")

    # Test EMBOSS
    print("1. Testing EMBOSS...")
    try:
        import subprocess
        result = subprocess.run(['embossversion'], capture_output=True,
text=True)
        print(f"   ✅ EMBOSS: {result.stdout.strip()}")
    except:
        print("   ❌ EMBOSS not found")

    # Test BioPython
    print("2. Testing BioPython...")
    try:
        from Bio import SeqIO
        print("   ✅ BioPython installed")
    except:
        print("   ❌ BioPython not found")

    # Test Ollama
    print("3. Testing Ollama...")
    try:
        import ollama
        client = ollama.Client()
        models = client.list()
        print(f"   ✅ Ollama running with {len(models['models'])} models")
    except:
        print("   ❌ Ollama not running or not installed")

    # Test the main application
    print("4. Testing BioQuery Local...")
    try:
        from bioquery_local import BioQueryLocal
        bq = BioQueryLocal()
        result = bq.process_query("Translate ATGGCG")
        if result['success']:
            print("   ✅ BioQuery Local working!")
        else:
            print(f"   ⚠️  BioQuery Local returned error: 
{result['error']}")
    except Exception as e:
        print(f"   ❌ BioQuery Local error: {e}")

if __name__ == "__main__":
    test_installation()
```

Setup Instructions for Students

None
~~~
# BioQuery Local Setup Guide

## 1. Install Conda (if not already installed)
Download from: https://docs.conda.io/en/latest/miniconda.html

## 2. Create Environment
```bash
conda create -n bioquery python=3.10
conda activate bioquery
~~~

3. Install Bioinformatics Tools

Shell
```
conda install -c bioconda emboss biopython
conda install -c conda-forge streamlit pandas
pip install ollama
```

4. Install Ollama (macOS)

Shell
```
# Using Homebrew
brew install ollama
```

```
# Start Ollama service
ollama serve

# In a new terminal, pull a model
ollama pull phi3:mini
```

5. Test Installation

Shell
```
python test_bioquery.py
```

6. Run the Application

Shell
```
streamlit run app.py
```

Your browser should open to http://localhost:8501

None
```
### **Advantages of This Local Approach**

1. **No API Keys Required**: Everything runs locally
2. **Educational Transparency**: Students can see exactly what's happening
3. **Consistent Results**: No API rate limits or downtime
4. **Privacy**: No data leaves the student's machine
5. **Real Tools**: Students learn actual bioinformatics tools (EMBOSS)
6. **Lightweight**: Phi-3 mini is only 2GB and runs well on modest hardware

### **Week-by-Week Deliverables**

- **Week 1**: Working EMBOSS wrapper with 3+ tools
- **Week 2**: LLM integration parsing queries correctly
- **Week 3**: BioPython tools added, test suite passing
- **Week 4**: Complete query processing pipeline
- **Week 5**: Web interface with documentation

This approach gives students hands-on experience with real bioinformatics
tools while teaching them about natural language processing and modern AI
integration, all without any cloud dependencies!
```