4-Person Team Division - Medical Imaging Hackathon

Team Structure & Responsibilities

Person 1: AWS Infrastructure & Backend Lead

Role: Cloud Infrastructure & API Development **Time Allocation:** Full 3 days

Person 2: AI/ML & SageMaker Specialist

Role: Machine Learning Models & AI Integration **Time Allocation:** Full 3 days

Person 3: IBM watsonx.ai & LLM Integration

Role: Large Language Model & Clinical Report Generation **Time Allocation:** Full 3 days

Person 4: Frontend & Integration Lead

Role: User Interface & System Integration **Time Allocation:** Full 3 days

---

PERSON 1: AWS Infrastructure & Backend Lead

Day 1 Tasks (Setup & Storage)

Morning (9 AM - 12 PM)

AWS Account & Basic Setup

```bash
# 1. Create AWS Account and configure CLI
aws configure
aws sts get-caller-identity  # Verify setup

# 2. Create S3 bucket for medical images
aws s3 mb s3://hackathon-medical-images-$(date +%s)
aws s3api put-bucket-versioning --bucket your-bucket --versioning-configuration Status=Enabled

# 3. Enable S3 event notifications
aws s3api put-bucket-notification-configuration \
  --bucket your-bucket \
  --notification-configuration file://s3-notification.json
```

Create s3-notification.json:

```json
```

```
{
 "LambdaConfigurations": [
  {
   "Id": "ProcessMedicalImage",
   "LambdaFunctionArn": "arn:aws:lambda:region:account:function:medical-processor",
   "Events": ["s3:ObjectCreated:*"],
   "Filter": {
    "Key": {
     "FilterRules": [
      {"Name": "suffix", "Value": ".dcm"},
      {"Name": "suffix", "Value": ".jpg"},
      {"Name": "suffix", "Value": ".png"}
     ]
    }
   }
  }
 ]
}
```

Afternoon (1 PM - 6 PM)

Lambda Functions Development

Main Processing Lambda (medical-processor):

```python
```

```python
import json
import boto3
import base64
import os
from datetime import datetime

s3_client = boto3.client('s3')
lambda_client = boto3.client('lambda')

def lambda_handler(event, context):
    try:
        # Extract S3 event information
        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']

        print(f"Processing image: {key} from bucket: {bucket}")

        # Get image metadata
        response = s3_client.head_object(Bucket=bucket, Key=key)
        image_size = response['ContentLength']

        # Create processing job
        job_data = {
            'bucket': bucket,
            'key': key,
            'timestamp': datetime.utcnow().isoformat(),
            'size': image_size,
            'status': 'processing'
        }

        # Store job info in S3
        job_key = f"jobs/{key.split('/')[-1]}.json"
        s3_client.put_object(
            Bucket=bucket,
            Key=job_key,
            Body=json.dumps(job_data),
            ContentType='application/json'
        )

        # Trigger AI analysis (invoke Person 2's function)
        lambda_client.invoke(
            FunctionName='ai-analysis-function',
            InvocationType='Event',
            Payload=json.dumps({
                'bucket': bucket,
                'image_key': key,
                'job_key': job_key
            })
        )

        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': 'Processing started',
                'job_id': job_key
            })
        }

    except Exception as e:
        print(f"Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

Deploy Lambda:

bash

```
# Create deployment package
zip -r medical-processor.zip lambda_function.py

# Create Lambda function
aws lambda create-function \
  --function-name medical-processor \
  --runtime python3.9 \
  --role arn:aws:iam::account:role/lambda-execution-role \
  --handler lambda_function.lambda_handler \
  --zip-file fileb://medical-processor.zip
```

Day 2 Tasks (API Gateway & Integration)

Morning (9 AM - 12 PM)

API Gateway Setup

bash

```
# Create REST API
aws apigateway create-rest-api --name medical-imaging-api

# Create resources and methods
# /upload endpoint for image uploads
# /status/{job_id} for checking processing status
# /results/{job_id} for getting final results
```

API Gateway Lambda Integration:

python

```
# upload-api-handler.py
import json
import boto3
import uuid
import base64

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    try:
        # Handle CORS
        headers = {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'POST, GET, OPTIONS'
        }

        if event['httpMethod'] == 'OPTIONS':
            return {
                'statusCode': 200,
                'headers': headers,
                'body': ''
            }

        # Handle file upload
        if event['httpMethod'] == 'POST':
            # Parse multipart form data or base64
            body = json.loads(event['body'])

            # Generate unique filename
            file_id = str(uuid.uuid4())
            filename = f"uploads/{file_id}.jpg"

            # Upload to S3
            s3_client.put_object(
                Bucket=os.environ['BUCKET_NAME'],
                Key=filename,
                Body=base64.b64decode(body['image']),
                ContentType='image/jpeg'
            )

            return {
                'statusCode': 200,
                'headers': headers,
                'body': json.dumps({
                    'job_id': file_id,
                    'status': 'uploaded',
                    'message': 'Image uploaded successfully'
                })
            }

        # Handle status check
        elif event['httpMethod'] == 'GET' and 'job_id' in event['pathParameters']:
            job_id = event['pathParameters']['job_id']

            # Check job status in S3
            try:
                response = s3_client.get_object(
                    Bucket=os.environ['BUCKET_NAME'],
                    Key=f"results/{job_id}.json"
                )
                result = json.loads(response['Body'].read())

                return {
                    'statusCode': 200,
                    'headers': headers,
                    'body': json.dumps(result)
                }
            except:
                return {
                    'statusCode': 202,
                    'headers': headers,
                    'body': json.dumps({'status': 'processing'})
                }

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': headers,
            'body': json.dumps({'error': str(e)})
        }
```

Afternoon (1 PM - 6 PM)

Monitoring & Error Handling

CloudWatch Setup:

```python
```

```python
# monitoring-function.py
import boto3
import json

cloudwatch = boto3.client('cloudwatch')

def put_custom_metric(metric_name, value, unit='Count'):
    cloudwatch.put_metric_data(
        Namespace='MedicalImaging',
        MetricData=[
            {
                'MetricName': metric_name,
                'Value': value,
                'Unit': unit
            }
        ]
    )

def lambda_handler(event, context):
    # Track processing times, success rates, etc.
    put_custom_metric('ImagesProcessed', 1)
    return {'statusCode': 200}
```

Day 3 Tasks (Final Integration & Testing)

Full Day (9 AM - 6 PM)

System Integration & Testing

1. **Connect all Lambda functions**
2. **Test end-to-end flow**
3. **Performance optimization**
4. **Error handling refinement**
5. **Documentation for team**

Final API Endpoints Documentation:

```
POST /upload - Upload medical image
GET /status/{job_id} - Check processing status
GET /results/{job_id} - Get analysis results
```

PERSON 2: AI/ML & SageMaker Specialist

Day 1 Tasks (Model Research & Setup)

Morning (9 AM - 12 PM)

SageMaker Environment Setup

```python
# setup-sagemaker.py
import boto3
import sagemaker
from sagemaker import get_execution_role

# Initialize SageMaker session
sagemaker_session = sagemaker.Session()
role = get_execution_role()  # Create this IAM role

# Create S3 bucket for models
bucket = sagemaker_session.default_bucket()
prefix = 'medical-imaging-models'

print(f"SageMaker role: {role}")
print(f"S3 bucket: {bucket}")
```

Create SageMaker Execution Role:

```bash
# IAM role for SageMaker
aws iam create-role --role-name SageMakerExecutionRole \
  --assume-role-policy-document file://trust-policy.json

aws iam attach-role-policy --role-name SageMakerExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonSageMakerFullAccess
```

Afternoon (1 PM - 6 PM)

Model Selection & Preparation

Option 1: Use Pre-trained HuggingFace Model

```python
```

```python
# medical-model-setup.py
from sagemaker.huggingface import HuggingFaceModel
import sagemaker

def deploy_medical_model():
    # Use a pre-trained medical imaging model
    huggingface_model = HuggingFaceModel(
        model_data="s3://huggingface-models/medical-imaging/",
        role=get_execution_role(),
        transformers_version="4.21",
        pytorch_version="1.12",
        py_version="py39",
        predictor_cls=sagemaker.predictor.Predictor
    )

    # Deploy to endpoint
    predictor = huggingface_model.deploy(
        initial_instance_count=1,
        instance_type="ml.m5.large",
        endpoint_name="medical-imaging-endpoint"
    )

    return predictor
```

Option 2: Quick Custom Model (Faster for hackathon)

python

```python
# simple-medical-classifier.py
import json
import torch
import torchvision.transforms as transforms
from PIL import Image
import boto3

class SimpleMedicalClassifier:
    def __init__(self):
        # Load pre-trained ResNet and adapt for medical imaging
        self.model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pretrained=True)
        self.model.eval()

        self.transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
        ])

        # Medical conditions mapping (simplified for demo)
        self.conditions = {
            0: "normal",
            1: "potential_abnormality",
            2: "urgent_finding"
        }

    def analyze_image(self, image_bytes):
        # Convert bytes to PIL Image
        image = Image.open(io.BytesIO(image_bytes))

        # Apply transforms
        input_tensor = self.transform(image).unsqueeze(0)

        # Get prediction
        with torch.no_grad():
            outputs = self.model(input_tensor)
            probabilities = torch.nn.functional.softmax(outputs[0], dim=0)

        # Return structured results
        return {
            "findings": self.conditions.get(torch.argmax(probabilities).item(), "unknown"),
            "confidence": float(torch.max(probabilities)),
            "all_probabilities": probabilities.tolist()[:3]  # Top 3
        }
```

```python
# Lambda function for AI analysis
def lambda_handler(event, context):
    s3_client = boto3.client('s3')
    classifier = SimpleMedicalClassifier()

    try:
        bucket = event['bucket']
        image_key = event['image_key']
        job_key = event['job_key']

        # Download image from S3
        response = s3_client.get_object(Bucket=bucket, Key=image_key)
        image_bytes = response['Body'].read()

        # Analyze image
        results = classifier.analyze_image(image_bytes)

        # Add metadata
        results['image_key'] = image_key
        results['processing_time'] = '2.3 seconds'  # Mock for demo
        results['model_version'] = 'v1.0'

        # Save results to S3
        result_key = f"ai-results/{job_key}"
        s3_client.put_object(
            Bucket=bucket,
            Key=result_key,
            Body=json.dumps(results),
            ContentType='application/json'
        )

        # Trigger LLM processing (Person 3's function)
        lambda_client = boto3.client('lambda')
        lambda_client.invoke(
            FunctionName='llm-processing-function',
            InvocationType='Event',
            Payload=json.dumps({
                'bucket': bucket,
                'ai_results_key': result_key,
                'job_key': job_key
            })
        )

        return {
            'statusCode': 200,
            'body': json.dumps(results)
        }

    except Exception as e:
        print(f"AI Analysis Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

Day 2 Tasks (Model Deployment & Testing)

Morning (9 AM - 12 PM)

Deploy & Test Models

bash

```bash
# Deploy Lambda function for AI processing
zip -r ai-analysis.zip lambda_function.py model_files/

aws lambda create-function \
  --function-name ai-analysis-function \
  --runtime python3.9 \
  --role arn:aws:iam::account:role/lambda-execution-role \
  --handler lambda_function.lambda_handler \
  --zip-file fileb://ai-analysis.zip \
  --timeout 300 \
  --memory-size 1024
```

Afternoon (1 PM - 6 PM)

Model Optimization & Validation

python

```
# model-validation.py
import json
import time

def validate_model_performance():
    test_cases = [
        "sample_ct_normal.jpg",
        "sample_ct_stroke.jpg",
        "sample_mri_hemorrhage.jpg"
    ]

    results = []

    for test_image in test_cases:
        start_time = time.time()

        # Test your model
        result = analyze_test_image(test_image)

        processing_time = time.time() - start_time

        results.append({
            'image': test_image,
            'result': result,
            'processing_time': processing_time
        })

    return results

def create_demo_dataset():
    """Create sample medical images with known results for demo"""
    sample_results = {
        "normal_ct": {
            "findings": "normal",
            "confidence": 0.92,
            "description": "No acute abnormalities detected"
        },
        "stroke_ct": {
            "findings": "urgent_finding",
            "confidence": 0.87,
            "description": "Possible acute stroke - left MCA territory"
        },
        "hemorrhage_ct": {
            "findings": "urgent_finding",
            "confidence": 0.94,
            "description": "Intracranial hemorrhage detected"
        }
    }

    return sample_results
```

Day 3 Tasks (Integration & Performance)

Full Day (9 AM - 6 PM)

Final Model Integration & Testing

1. **Integration testing with Person 1's infrastructure**
2. **Performance optimization**
3. **Mock realistic medical results for demo**
4. **Coordinate with Person 3 for LLM handoff**

PERSON 3: IBM watsonx.ai & LLM Integration

Day 1 Tasks (IBM Setup & LLM Access)

Morning (9 AM - 12 PM)

IBM Cloud Setup

```bash
# Install IBM Cloud CLI
curl -fsSL https://clis.cloud.ibm.com/install/linux | sh

# Login and setup
ibmcloud login --apikey YOUR_API_KEY
ibmcloud target -r us-south -g default
```

Python Setup for watsonx.ai:

```python
```

```
# watson-setup.py
from ibm_watson_machine_learning import APIClient
import json

# Watson ML credentials
wml_credentials = {
    "url": "https://us-south.ml.cloud.ibm.com",
    "apikey": "your-api-key-here",
    "instance_id": "your-instance-id"
}

# Initialize client
client = APIClient(wml_credentials)

# Set project
client.set.default_project("your-project-id")

# List available foundation models
models = client.foundation_models.get_model_specs()
print("Available models:", models)

# Test connection
print("Watson ML client initialized successfully!")
```

Afternoon (1 PM - 6 PM)

LLM Prompt Engineering

python

```
# medical-prompt-templates.py
class MedicalPromptTemplates:

    @staticmethod
    def clinical_summary_prompt(ai_findings, patient_context="emergency"):
        return f"""
You are an expert radiologist AI assistant. Based on the medical imaging analysis results, generate a concise

IMAGING FINDINGS:
{ai_findings}

CLINICAL CONTEXT: {patient_context}

Please provide:
1. KEY FINDINGS: (2-3 sentences max)
2. CLINICAL SIGNIFICANCE: (Critical/Moderate/Low concern)
3. RECOMMENDED ACTIONS: (Immediate steps)
4. FOLLOW-UP: (If needed)

Keep the language clear and actionable for emergency physicians. Focus on time-sensitive information.

CLINICAL SUMMARY:
"""

    @staticmethod
    def treatment_protocol_prompt(findings, condition):
        return f"""
Based on the medical imaging findings showing {condition}, provide the appropriate emergency treatment pr

FINDINGS: {findings}

Provide a structured treatment protocol including:
- IMMEDIATE ACTIONS (within 15 minutes)
- DIAGNOSTIC WORKUP (additional tests needed)
- TREATMENT OPTIONS (evidence-based)
- CONSULTATION REQUIREMENTS (specialist referrals)

TREATMENT PROTOCOL:
"""

    @staticmethod
    def patient_explanation_prompt(technical_findings):
        return f"""
Convert the following technical medical findings into simple language for patient/family explanation:

TECHNICAL FINDINGS: {technical_findings}
```

```
Provide a clear, compassionate explanation that:
- Uses simple, non-technical language
- Explains what was found
- Explains next steps
- Addresses likely concerns

PATIENT EXPLANATION:
"""

# Test the prompts
def test_prompts():
    sample_findings = {
        "findings": "urgent_finding",
        "confidence": 0.87,
        "description": "Possible acute stroke - left MCA territory"
    }

    prompt = MedicalPromptTemplates.clinical_summary_prompt(
        json.dumps(sample_findings),
        "emergency"
    )

    print("Generated Prompt:")
    print(prompt)
```

Day 2 Tasks (LLM Integration & Processing)

Morning (9 AM - 12 PM)

Granite LLM Integration

python

```
# llm-processor.py
from ibm_watson_machine_learning import APIClient
import json
import boto3

class GraniteMedicalLLM:
    def __init__(self):
        self.wml_credentials = {
            "url": "https://us-south.ml.cloud.ibm.com",
            "apikey": os.environ['IBM_API_KEY'],
            "instance_id": os.environ['IBM_INSTANCE_ID']
        }

        self.client = APIClient(self.wml_credentials)
        self.client.set.default_project(os.environ['IBM_PROJECT_ID'])

        # Model parameters
        self.generation_params = {
            "max_new_tokens": 300,
            "temperature": 0.3,
            "top_p": 0.9,
            "repetition_penalty": 1.1
        }

    def generate_clinical_summary(self, ai_findings):
        """Generate clinical summary using Granite LLM"""
        try:
            prompt = MedicalPromptTemplates.clinical_summary_prompt(ai_findings)

            # Generate response
            response = self.client.foundation_models.generate_text(
                model_id="ibm/granite-13b-chat-v2",  # or latest Granite model
                prompt=prompt,
                params=self.generation_params
            )

            return {
                "clinical_summary": response,
                "model_used": "granite-13b-chat-v2",
                "confidence": "high",
                "generated_at": datetime.utcnow().isoformat()
            }

        except Exception as e:
            print(f"LLM Generation Error: {str(e)}")
            # Fallback to template-based response
            return self.fallback_summary(ai_findings)

    def fallback_summary(self, ai_findings):
        """Fallback summary in case LLM fails"""
        findings_data = json.loads(ai_findings) if isinstance(ai_findings, str) else ai_findings

        if findings_data.get('findings') == 'urgent_finding':
            return {
                "clinical_summary": """
KEY FINDINGS: Potential urgent abnormality detected on imaging with high confidence.
CLINICAL SIGNIFICANCE: HIGH CONCERN - requires immediate physician review.
RECOMMENDED ACTIONS:
- Immediate physician evaluation
- Consider specialist consultation
- Monitor patient closely
FOLLOW-UP: Formal radiologist interpretation recommended within 1 hour.
                """,
                "model_used": "fallback_template",
                "confidence": "template_based"
            }
        else:
            return {
                "clinical_summary": """
KEY FINDINGS: No acute abnormalities detected on initial AI screening.
CLINICAL SIGNIFICANCE: LOW CONCERN - routine findings.
RECOMMENDED ACTIONS:
- Continue standard clinical evaluation
- Formal radiologist review as scheduled
FOLLOW-UP: Standard radiology reporting timeframe.
                """,
                "model_used": "fallback_template",
                "confidence": "template_based"
            }

# Lambda function for LLM processing
def lambda_handler(event, context):
    s3_client = boto3.client('s3')
    llm_processor = GraniteMedicalLLM()

    try:
        bucket = event['bucket']
        ai_results_key = event['ai_results_key']
        job_key = event['job_key']

        # Get AI analysis results from S3
        response = s3_client.get_object(Bucket=bucket, Key=ai_results_key)
        ai_results = json.loads(response['Body'].read())
        # Generate clinical summary
        clinical_summary = llm_processor.generate_clinical_summary(ai_results)

        # Combine results
        final_results = {
            "ai_analysis": ai_results,
            "clinical_summary": clinical_summary,
            "processing_complete": True,
            "total_processing_time": "2.8 seconds",  # Mock for demo
            "timestamp": datetime.utcnow().isoformat()
        }

        # Save final results to S3
        final_key = f"results/{job_key.replace('jobs/', '').replace('.json', '')}.json"
        s3_client.put_object(
            Bucket=bucket,
            Key=final_key,
            Body=json.dumps(final_results),
            ContentType='application/json'
        )

        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': 'Clinical summary generated',
                'results_key': final_key
            })
        }

    except Exception as e:
        print(f"LLM Processing Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

Afternoon (1 PM - 6 PM)

Medical Knowledge Base

python

```python
# medical-guidelines.py
class MedicalGuidelinesDB:
    def __init__(self):
        self.guidelines = {
            "stroke": {
                "protocol": "Activate stroke code immediately",
                "time_window": "4.5 hours for thrombolysis",
                "imaging": "Non-contrast CT first, then CT angiography",
                "treatment": "Consider tPA if within window"
            },
            "hemorrhage": {
                "protocol": "Neurosurgery consultation stat",
                "imaging": "Non-contrast CT diagnostic",
                "treatment": "Reverse anticoagulation if present",
                "monitoring": "Neuro checks q15min"
            },
            "normal": {
                "protocol": "Standard emergency evaluation",
                "next_steps": "Clinical correlation recommended",
                "follow_up": "Routine radiology review"
            }
        }

    def get_guideline(self, condition):
        return self.guidelines.get(condition.lower(), self.guidelines["normal"])

    def enhance_summary_with_guidelines(self, summary, condition):
        guideline = self.get_guideline(condition)

        enhanced = f"""
{summary}

CLINICAL GUIDELINES:
Protocol: {guideline.get('protocol', 'Standard care')}
Timing: {guideline.get('time_window', 'No specific time constraints')}
Next Steps: {guideline.get('treatment', 'Continue clinical evaluation')}
        """

        return enhanced
```

Day 3 Tasks (Testing & Optimization)

Full Day (9 AM - 6 PM)

LLM Testing & Integration

1. **Test different prompt variations**
2. **Optimize response times**
3. **Create fallback mechanisms**
4. **Integration testing with team**

PERSON 4: Frontend & Integration Lead

Day 1 Tasks (Frontend Setup)

Morning (9 AM - 12 PM)

Basic HTML/CSS/JS Setup

```html
```

```html
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Emergency Medical Imaging AI</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/tailwindcss/2.2.19/tailwind.min.css" rel="stylesheet">
    <style>
        .upload-area {
            border: 2px dashed #cbd5e0;
            transition: all 0.3s ease;
        }
        .upload-area.dragover {
            border-color: #3182ce;
            background-color: #ebf8ff;
        }
        .processing {
            animation: pulse 2s infinite;
        }
    </style>
</head>
<body class="bg-gray-100">
    <div class="container mx-auto px-4 py-8">
        <h1 class="text-4xl font-bold text-center text-gray-800 mb-8">
            🏥 Emergency Medical Imaging AI
        </h1>

        <!-- Upload Section -->
        <div class="max-w-2xl mx-auto">
            <div id="uploadArea" class="upload-area rounded-lg p-8 text-center mb-6">
                <div id="uploadContent">
                    <svg class="mx-auto h-12 w-12 text-gray-400 mb-4" stroke="currentColor" fill="none" viewBox
                        <path d="M28 8H12a4 4 0 00-4 4v20m32-12v8m0 0v8a4 4 0 01-4 4H12a4 4 0 01-4-4v-4m3
                    </svg>
                    <p class="text-xl text-gray-600 mb-4">Upload CT or MRI Scan</p>
                    <p class="text-gray-500 mb-4">Drag and drop or click to select</p>
                    <input type="file" id="imageInput" accept="image/*,.dcm" class="hidden">
                    <button onclick="document.getElementById('imageInput').click()"
                            class="bg-blue-500 hover:bg-blue-600 text-white px-6 py-3 rounded-lg">
                        Select Image
                    </button>
                </div>

                <!-- Processing State -->
                <div id="processingState" class="hidden">
            <div class="processing">
              <div class="inline-block w-8 h-8 border-4 border-blue-500 border-t-transparent rounded-full
            </div>
            <p class="text-xl text-blue-600 mb-2">Analyzing Medical Image...</p>
            <p id="processingStatus" class="text-gray-600">Initializing AI analysis</p>
            <div class="bg-gray-200 rounded-full h-2 mt-4">
              <div id="progressBar" class="bg-blue-500 h-2 rounded-full transition-all duration-500" style=
            </div>
          </div>
        </div>

        <!-- Results Section -->
        <div id="resultsSection" class="hidden">
          <div class="bg-white rounded-lg shadow-lg p-6">
            <h2 class="text-2xl font-bold text-gray-800 mb-4">Analysis Results</h2>

            <!-- AI Findings -->
            <div class="mb-6">
              <h3 class="text-lg font-semibold text-gray-700 mb-2">AI Detection Results</h3>
              <div id="aiFindings" class="bg-gray-50 rounded p-4">
                <!-- Dynamic content -->
```