TDIU Report Service: Claude AI Integration via AWS Bedrock

Overview

The Claude AI integration is a core component of the TDIU Report Service, providing intelligent document analysis capabilities for veteran medical records, employment histories, and other claim-related documents. This integration leverages AWS Bedrock to access Claude 3.7 Sonnet in a HIPAA-compliant manner, ensuring both powerful AI capabilities and enterprise-grade security.

Key Capabilities

The Claude AI integration provides several critical capabilities:

1. **Medical Document Analysis**
   - Identify medical conditions and diagnoses
   - Extract symptoms and limitations
   - Categorize treatments and medications
   - Connect conditions to functional limitations
   - Identify relevant medical evidence for TDIU claims
2. **Employment History Analysis**
   - Categorize past employment by skill level and requirements
   - Identify accommodations and limitations
   - Extract workplace challenges related to disabilities
   - Analyze employment gaps and patterns
   - Connect medical limitations to employment difficulties
3. **Report Generation Assistance**
   - Structure findings in VA-compatible formats
   - Generate draft sections for reports
   - Ensure comprehensive coverage of all claim elements
   - Maintain consistent professional language
   - Suggest evidence connections and relevant VA regulations
4. **Multi-Service Support**
   - Flexible prompting for different service needs
   - Specialized prompts for each service type

- Consistent output formatting across services
- Knowledge sharing between service types
- Efficient resource utilization

AWS Bedrock Implementation

1. Technical Architecture

AWS Bedrock provides a managed service for accessing foundation models including Claude 3.7 Sonnet. The implementation includes:

1. **Lambda Integration**
   - TDIU-DocumentProcessor Lambda function connects to Bedrock
   - S3 event triggers document processing
   - Secure handling of document content
   - Proper error handling and retry logic
2. **Security Configuration**
   - IAM roles with minimal necessary permissions
   - Encryption for all data in transit
   - Temporary credential management
   - Audit logging of all operations
3. **Resource Management**
   - Efficient token usage to control costs
   - Batching for optimal processing
   - Caching where appropriate
   - Asynchronous processing for large documents

2. Lambda Implementation

The TDIU-DocumentProcessor Lambda function will include:

python

```python
import boto3
import json
import os
from urllib.parse import unquote_plus

s3_client = boto3.client('s3')
bedrock_runtime = boto3.client('bedrock-runtime')

def handler(event, context):
    # Get the uploaded file info from the S3 event
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])

        # Get the document content
        response = s3_client.get_object(Bucket=bucket, Key=key)
        document_content = response['Body'].read().decode('utf-8')

        # Prepare document for analysis
        document_type = determine_document_type(key)
        prompt = generate_prompt(document_type, document_content)

        # Call Claude AI via Bedrock
        analysis_result = analyze_document(prompt)

        # Save the analysis result
        result_key = f"analysis/{key.split('/')[-1]}_analysis.json"
        s3_client.put_object(
            Bucket=bucket,
            Key=result_key,
            Body=json.dumps(analysis_result),
            ContentType='application/json',
            ServerSideEncryption='AES256'
        )

        return {
            'statusCode': 200,
            'body': json.dumps({'message': 'Document processed successfully', 'result': result_
        }

def determine_document_type(key):
    # Logic to determine if this is a medical record, employment history, etc.
    if 'medical' in key.lower():
```

```
        return 'medical'
    elif 'employment' in key.lower():
        return 'employment'
    else:
        return 'general'

def generate_prompt(document_type, content):
    # Generate appropriate prompt based on document type
    prompts = {
        'medical': """
            You are analyzing a medical document for a TDIU claim.
            Please identify:
            1. All medical conditions and diagnoses
            2. Symptoms and functional limitations
            3. Treatments and medications
            4. Relevant evidence for TDIU eligibility

            Document content:
            {content}

            Provide your analysis in structured JSON format.
        """,
        'employment': """
            You are analyzing an employment history document for a TDIU claim.
            Please identify:
            1. Employment timeline and gaps
            2. Job requirements and skill levels
            3. Accommodations and limitations
            4. Evidence of inability to maintain gainful employment

            Document content:
            {content}

            Provide your analysis in structured JSON format.
        """,
        'general': """
            You are analyzing a document for a TDIU claim.
            Please identify any relevant information related to:
            1. Medical conditions
            2. Employment history
            3. Functional limitations
            4. Educational background
            5. Evidence supporting TDIU eligibility
```

```
            Document content:
            {content}

            Provide your analysis in structured JSON format.
        """
    }

    selected_prompt = prompts.get(document_type, prompts['general'])
    return selected_prompt.format(content=content)

def analyze_document(prompt):
    # Call Bedrock with Claude 3.7 Sonnet
    response = bedrock_runtime.invoke_model(
        modelId='anthropic.claude-3-sonnet-20240229-v1:0',
        contentType='application/json',
        accept='application/json',
        body=json.dumps({
            "anthropic_version": "bedrock-2023-05-31",
            "max_tokens": 4096,
            "temperature": 0.0,
            "messages": [
                {
                    "role": "user",
                    "content": prompt
                }
            ]
        })
    )

    response_body = json.loads(response['body'].read())
    ai_response = response_body['content'][0]['text']

    # Extract structured data from response
    try:
        # Check if response is already JSON
        if ai_response.strip().startswith('{'):
            analysis_data = json.loads(ai_response)
        else:
            # Extract JSON data if embedded in text
            analysis_data = {
                'raw_analysis': ai_response,
                'structured_data': extract_structured_data(ai_response)
            }
    except json.JSONDecodeError:
```

```
            analysis_data = {'raw_analysis': ai_response}

        return analysis_data

    def extract_structured_data(text):
        # Attempt to extract structured data from text
        # This is a placeholder - actual implementation would be more robust
        structured_data = {
            'conditions': [],
            'limitations': [],
            'treatments': [],
            'employment_factors': []
        }

        # Simple extraction logic
        if 'diagnos' in text.lower():
            structured_data['conditions'] = ['Found potential diagnosis mentions']

        return structured_data
```

3. CloudFormation Template Update

The following additions to the CloudFormation template will implement the Claude AI integration:

yaml

```yaml
# Claude AI Integration Resources
DocumentProcessorFunction:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: TDIU-DocumentProcessor
    Runtime: python3.9
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    Timeout: 60
    MemorySize: 512
    Environment:
      Variables:
        RESULTS_BUCKET: !Ref DocumentStorageBucket
    Code:
      ZipFile: |
        # Lambda code would go here (simplified for template)
        def handler(event, context):
            return {'statusCode': 200, 'body': 'Processing complete'}

DocumentUploadTrigger:
  Type: AWS::Lambda::Permission
  Properties:
    Action: lambda:InvokeFunction
    FunctionName: !GetAtt DocumentProcessorFunction.Arn
    Principal: s3.amazonaws.com
    SourceArn: !GetAtt DocumentStorageBucket.Arn

DocumentProcessingEvent:
  Type: AWS::S3::BucketNotification
  Properties:
    Bucket: !Ref DocumentStorageBucket
    NotificationConfiguration:
      LambdaConfigurations:
        - Event: s3:ObjectCreated:*
          Filter:
            S3Key:
              Rules:
                - Name: prefix
                  Value: uploads/
          Function: !GetAtt DocumentProcessorFunction.Arn
```

Prompt Engineering for TDIU Reports

Effective prompt engineering is critical for accurate document analysis and report generation. The system will use specialized prompts for different document types and processing stages:

1. Medical Record Analysis Prompt

```
You are a medical document analyzer for TDIU (Total Disability Individual Unemployability)
claims. You are reviewing medical records for a veteran seeking VA disability benefits based
on unemployability.

Please analyze the following medical record excerpt:

[DOCUMENT_CONTENT]

For each medical condition mentioned, provide:
1. Condition name and diagnosis codes (if available)
2. Date of diagnosis or documentation
3. Treating provider information
4. Symptoms and severity indicators
5. Functional limitations described (especially related to employment)
6. Treatments prescribed and their effectiveness
7. Prognosis or expected duration
8. Direct quotes from the record that support TDIU eligibility

Format your response as structured JSON with fields for each category. Include a
"relevance_score" from 1-10 for each condition's impact on employability.
```

2. Employment History Analysis Prompt

```
You are an employment history analyzer for TDIU (Total Disability Individual
Unemployability) claims. You are reviewing employment records for a veteran seeking VA
disability benefits based on unemployability.

Please analyze the following employment document:

[DOCUMENT_CONTENT]

Provide the following analysis:
1. Complete employment timeline (with dates, employers, positions, and salary if available)
2. Job requirements for each position (physical, cognitive, educational)
3. Evidence of accommodations or limitations in the workplace
4. Pattern of decreasing hours, earnings, or responsibilities over time
5. Gaps in employment and stated reasons
6. Terminations, resignations, or job changes related to disability
7. Direct quotes that support inability to maintain substantially gainful employment

Format your response as structured JSON with sections for each employment period. For each
position, include a "sustainability_assessment" indicating whether the veteran could likely
continue this type of work with their documented conditions.
```

3. TDIU Report Generation Prompt

```
You are assisting in generating a TDIU (Total Disability Individual Unemployability) report for a veteran's attorney. You have analyzed medical and employment records and now need to generate structured sections for the final report.

Based on the following analysis data:

[ANALYSIS_JSON]

Please generate the following report sections:
1. Medical Evidence Summary (2-3 paragraphs describing key conditions and supporting evidence)
2. Vocational Impact Analysis (2-3 paragraphs on how conditions affect employment capability)
3. Employment History Assessment (1-2 paragraphs on past work and inability to continue)
4. Educational Factors (1 paragraph on educational background and transferable skills)
5. Key Limitations (bulleted list of functional limitations with citation to supporting evidence)
6. VA Criteria Application (1-2 paragraphs applying VA regulations to the evidence)
7. Conclusion (1 paragraph summarizing eligibility for TDIU)

Format your response in professional language appropriate for a legal document. Include specific citations to evidence where appropriate using this format: (Source: Document Name, Page X).
```

Multi-Service AI Implementation

The Claude AI integration is designed to support multiple services beyond the core TDIU report generation:

1. Document Analysis Service

Additional prompt templates will be created for the Document Analysis Service:

```
You are analyzing a comprehensive set of medical records to create an organized, searchable
document library. Please:

1. Create a chronological timeline of all medical visits and events
2. Categorize each document by type (progress note, lab result, imaging, etc.)
3. Identify all healthcare providers and their specialties
4. Extract all diagnoses and link to supporting evidence
5. Create a medication history with dosages and dates
6. Identify key evidence related to functional limitations
7. Cross-reference related documents

Format your analysis as structured JSON that can be used to organize a comprehensive medical
record library.
```

2. VA Brief Templates

Specialized prompts for the VA Brief Templates service:

```
You are customizing a VA brief template for a specific case. The template is for a
[TEMPLATE_TYPE] argument. Given the following case information:

[CASE_DETAILS]

Please generate appropriate language for each section of the template:
1. Introduction: Describe the specific issue and requested relief
2. Factual Background: Summarize relevant facts for this specific argument
3. Legal Standard: Adapt the standard legal language to this specific case
4. Analysis: Apply the facts to the legal standard for this case
5. Conclusion: State the requested outcome based on the analysis

Ensure the language is professional, persuasive, and specific to this veteran's
circumstances while following the template structure.
```

3. Medical Terms Translation

Specialized prompts for the Medical Terms Translation service:

```
You are translating complex medical terminology into plain language that clearly connects to
VA disability criteria. For the following medical terms:

[MEDICAL_TERMS]

For each term provide:
1. Plain language explanation
2. Functional impact description
3. Relevance to VA disability criteria
4. Example of how this would affect daily activities
5. Example of how this would affect work capabilities

Format your response as structured JSON with sections for each medical term. Use language
appropriate for a non-medical audience while maintaining clinical accuracy.
```

Monitoring and Optimization

The Claude AI integration will include comprehensive monitoring and optimization:

1. **Usage Monitoring**
   - Track token usage per document
   - Monitor processing time
   - Record error rates and types
   - Analyze response quality
2. **Cost Optimization**
   - Implement batching for multiple documents
   - Use chunking for large documents
   - Optimize prompt design for token efficiency
   - Implement caching where appropriate
3. **Quality Improvement**
   - Regular prompt refinement
   - Sample-based human review
   - Feedback loop for error correction
   - Version control for prompts
4. **Performance Metrics**
   - Processing time per document

- Cost per document
- Accuracy rates
- Human intervention rates

CloudFormation Integration

The complete Claude AI integration will be documented in the CloudFormation template, ensuring it can be fully reconstructed and understood:

yaml

```yaml
# Claude AI Integration Resources
BedrockPolicy:
  Type: AWS::IAM::ManagedPolicy
  Properties:
    ManagedPolicyName: TDIU-BedrockPolicy
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - bedrock:InvokeModel
          Resource: '*'

DocumentProcessorFunction:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: TDIU-DocumentProcessor
    Runtime: python3.9
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    Timeout: 60
    MemorySize: 512
    Environment:
      Variables:
        RESULTS_BUCKET: !Ref DocumentStorageBucket
    Code:
      ZipFile: |
        # Lambda code would go here (full implementation)

LambdaExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: TDIU-LambdaExecutionRole
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

```
      - arn:aws:iam::aws:policy/AmazonS3FullAccess
      - !Ref BedrockPolicy
```

Next Steps

1. **Complete Lambda Implementation**
   - Finalize the TDIU-DocumentProcessor code
   - Implement error handling and retry logic
   - Add logging for monitoring and debugging
2. **Test with Sample Documents**
   - Upload sample medical records
   - Test employment history processing
   - Verify combined document analysis
3. **Refine Prompts**
   - Optimize prompt structure for accuracy
   - Tune prompts for efficient token usage
   - Create specialized prompts for different document types
4. **Integrate with Report Generation**
   - Connect document analysis to report templates
   - Implement human review workflow
   - Develop quality assurance process
5. **Performance Testing**
   - Measure processing time for various document types
   - Calculate cost per document
   - Optimize for efficiency and cost effectiveness