Efficient Underwriting using Agentic AI

Mohammad Asif Ali
Technical Lead, PNC Financial Services Group, Inc, Pittsburgh, PA, USA
Email: asifali.ks@gmail.com

Abstract

In the financial industry, the underwriting process is an essential yet often protracted element of risk assessment. Traditional methods of underwriting are largely reliant on human expertise, rule-based evaluations, and statistical models, which can result in inefficiencies, inconsistencies, and delays in processing. This paper examines the groundbreaking implementation of **Agentic Artificial Intelligence (AI)** in underwriting, employing **large language models (LLMs)**, **retrieval-augmented generation (RAG), and robotic process automation (RPA)** to automate and refine decision-making processes. Through empirical validation, we demonstrate that Agentic AI significantly enhances the efficiency of loan processing, reduces bias, and improves the precision of risk assessments. Furthermore, the study compares the efficacy of AI-driven underwriting models against conventional methods, highlighting substantial advancements in processing speed, cost efficiency, and consistency in decision-making. Finally, we explore the challenges related to AI explainability, adherence to regulatory standards, and future prospects for AI-enhanced underwriting.

Keywords: Agentic AI, Artificial Intelligence, Loan Underwriter, Large Language Model(LLM), Automation, Retrieval Augmented Generation (RAG), Risk Assessment

Contents

1. Introduction

The financial sector, especially in the realm of loan underwriting, encounters considerable obstacles stemming from inefficiencies in manual processing, subjective risk evaluations, and increasing compliance requirements. Conventional underwriting practices typically rely on manual verification of documents, credit assessments, and human judgment, which can lead to inconsistencies and delays. The introduction of Agentic AI presents a revolutionary solution by incorporating AI-driven automation, intelligent decision-making capabilities, and real-time adaptability to enhance the efficiency of underwriting processes.

1.1. Research Objective

The objective of this paper is to:

1. Examine the influence of Agentic AI on the automation of underwriting processes.
2. Compare AI-driven underwriting with traditional approaches regarding accuracy, processing time, and risk profiling.
3. Analyze the contribution of Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) in the decision-making process of underwriting.
4. Explore the aspects of AI explainability, compliance, and future improvements.

1.2. Background of Agentic AI

Agentic AI is an emerging technology that is revolutionizing a wide range of industries. In order to create autonomous AI agents that can analyze data, set goals, and take actions with less human supervision, it combines enterprise automation, machine learning, and large language models (LLMs). With each interaction, these agents can make decisions, solve problems dynamically, learn, and get better.

With its constant learning and output optimization, Agentic AI is ideal for dynamic environments. In contrast to AI applications, which are frequently task-specific and excel in specialized fields like data analysis or image recognition, Agentic AI manages intricate, multi-step workflows that require real-time contextual understanding and decision-making.

AI agents, RPA robots, and humans work together in a symbiotic relationship in agentic automation. People give the agents their objectives, maintain governance, and intervene when human review and judgment are needed (human in the loop). By gathering the data needed for AI agents to make decisions (such as logging in, connecting, and comprehending information across multiple systems), RPA robots make AI agents more accurate, productive, and successful. They can also carry out a variety of other predetermined tasks for agents.

1.3. Agentic AI Levels

The concept of Agentic AI includes multiple levels, with each level indicating a varying degree of independence and intricacy in the actions and learning capabilities of AI systems.

1. **Reactive Agents**: Operating under a fundamental sense-and-respond paradigm, reactive agents respond to present inputs without the ability to retain information or engage in planning.
2. **Proactive Agents**: Proactive agents surpass the act of responding by utilizing lessons learned from past experiences and formulating targeted actions to reach their goals.
3. **Adaptive Agents**: Through dynamic learning from their interactions, adaptive agents enhance their responses and strategies, allowing them to operate efficiently within complex environments.
4. **Fully Agentic Systems**: Fully independent AI systems are designed to autonomously determine their objectives, make choices, and perform activities in environments that are not structured.

1.4. Future of Agentic AI

With the advancement of Agentic AI, its possibilities will broaden into more complex applications:

1. **Advanced Enterprise Autonomy:** AI agents that go beyond mere execution to include strategic planning and process optimization.
2. **Cross-Disciplinary Integration:** Merging domains like banking and finance, healthcare and legal research to achieve more profound solutions to challenges.
3. **Collaborative Human Interaction:** AI functioning as an active collaborator, working in harmony with human teams.

2. Literature Review

2.1 Traditional Underwriting vs. AI-Driven Underwriting

Traditional underwriting methods are based on rule-based decision-making, statistical analysis, and human oversight. Conversely, AI-enhanced underwriting utilizes machine learning (ML) models, natural language processing (NLP), and automation to boost efficiency and accuracy. Studies have demonstrated that AI underwriting can shorten loan approval times by 40-60% and increase the consistency of risk assessments by 30%.

2.2 Role of Large Language Models (LLMs) in Underwriting

The use of Large Language Models (LLMs), like Mistral 7B, is on the rise in areas such as document analysis, information extraction, and automated support for decision-making. By leveraging Retrieval-

Augmented Generation (RAG), these models facilitate adherence to underwriting policies and regulatory requirements during the decision-making process.

2.3 AI Bias, Explainability, and Compliance in Financial Decision-Making

One of the primary challenges associated with AI-driven underwriting is the presence of bias in model predictions, which can result from imbalances in the training datasets. Research underscores the necessity for interpretable AI models to comply with Fair Lending Practices and the Equal Credit Opportunity Acts. This study examines mitigation approaches, such as human-in-the-loop (HITL) oversight and federated learning, aimed at securing credit assessments.

2.1. Problem Statement

The high operational cost and the practice of traditional underwriting in the financial services industry has been reliant on human expertise, rule-based evaluations, and statistical models to assess the ability to pay the borrowed funds, determine loan eligibility, and evaluate insurance risks. While human underwriters provide critical domain knowledge, judgment, and ethical considerations, they are limited by factors such as subjectivity, inefficiency, extended processing times, and scalability challenges. Additionally, human-driven underwriting is at risk of biases, inconsistencies, and fatigue, which can negatively influence financial inclusion and risk management strategies.

With the advent of Agentic AI, there exists a promising opportunity to develop autonomous, adaptive, and self-improving underwriting agents that can carry out real-time risk assessments with superior efficiency, consistency, and scalability compared to traditional human underwriters.

2.2. Why Choose an Agentic AI Underwriter

The Agentic AI Underwriter has a very strong framework that leverages retrieval augmented generation (RAG) with large language models (LLMs) for faster indexation and knowledge retrieval. By harnessing Mistral AI services (Mistral 7B and MoE 8x7B), it creates a robust knowledge-driven AI agent which can generate intelligent suggestions, handle complex analysis and decision making. The data can be sourced from historical loan data, API integration with financial data sources, websites, policy documents, regulatory guidelines.

By simulating human judgment, Agentic AI Underwriter reduces the human intervention. AI agents prioritize tasks, allocate resources, and predict outcomes—implementing the decisions they make to move the process forward and achieve the desired outcome. It addresses the discrepancies like missing data or unexpected formats without human intervention.

Advantages:

1. **Enhanced efficiency and accuracy:** The conventional underwriting process is characterized by its labor-intensive nature and susceptibility to human error. By automating the phases of data collection and analysis, the AI agent greatly enhances the efficiency of underwriting, enabling human underwriters to evaluate applications at an unparalleled speed while maintaining the integrity of risk assessment.

2. **Automated Risk Profiling:** The Agentic AI underwriter significantly influences the assessment of risk factors. By utilizing AI platforms, the companies can swiftly gather and evaluate extensive datasets, including personal information and financial records, which enables them to develop a thorough understanding of an applicant's risk profile.
3. **Uniformity in Underwriting Practices:** The Agentic AI underwriter also promotes the uniformity of underwriting standards, guaranteeing that each application is assessed according to identical criteria. This consistency not only improves the equity of the underwriting process but also reduces the likelihood of human bias.
4. **Reduces operational cost:** The advantages of using the Agentic Ai underwriter go beyond the improvements in efficiency and accuracy. It helps to decrease the operational cost by reducing the extensive manual intervention. Additionally, the improved precision in risk evaluations creates a more competitive, equitable pricing and robust environment.

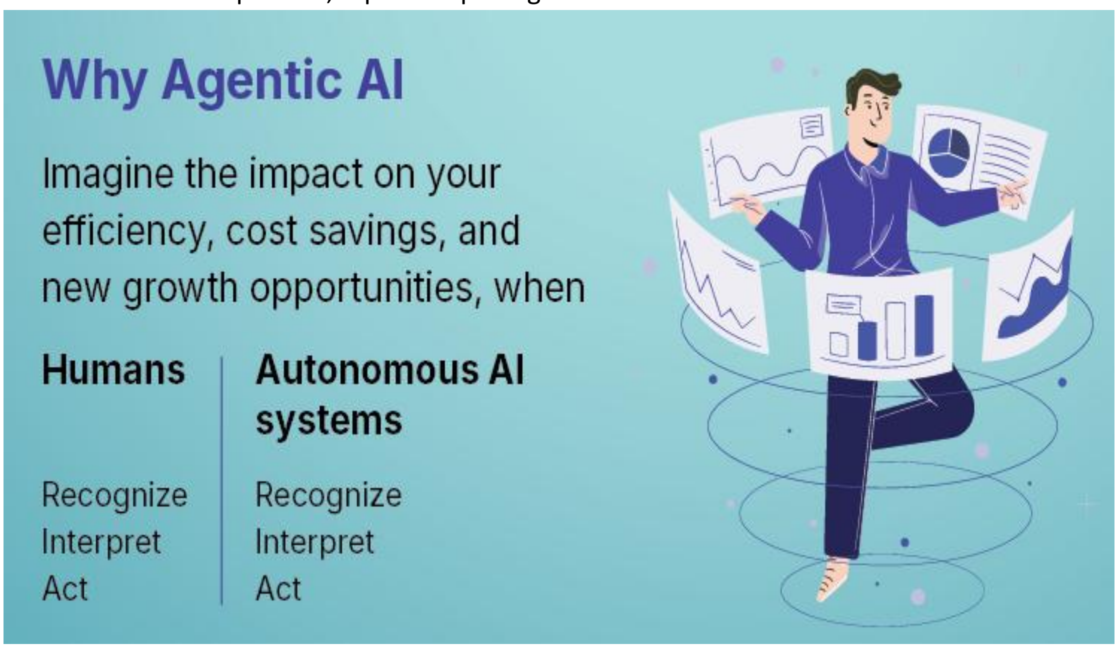


Figure 1. Human vs Agentic AI

3. Methodology

The Agentic AI Underwriter methodology integrates robotic process automation (RPA), sophisticated artificial intelligence (AI), and human oversight to enhance and refine the loan underwriting process. The procedure commences with the gathering of applicant information through document submissions and email interactions. A software robot is responsible for downloading these documents and forwarding them to Mayan EDMS for data extraction, utilizing the Mistral 7B large language model (LLM) to process and extract essential details such as the applicant's name, address, and income.

Concurrently, the bot accesses the applicant's credit score from credit bureaus via APIs. Additionally, it analyzes the sales representative’s email to extract pertinent loan information. This collected data is then input into the Mistral 7B Loan Recommendation AI Agent, which assesses the loan application in accordance with the bank's policies and guidelines stored within a Retrieval Augmented Generation (RAG) system, producing a recommendation based on compliance and risk evaluation.

The decision-making process bifurcates into two outcomes: approval or denial. In either scenario, a human-in-the-loop task enables a loan underwriter to review and confirm the AI-generated recommendation. Following the final decision, the bank's Loan Management System is updated through API, and automated email notifications are dispatched to all relevant parties, ensuring transparency and effective communication.

By merging automation, AI, and human oversight, this methodology significantly improves efficiency, shortens processing times, and upholds compliance, resulting in a robust and scalable loan underwriting solution.

4. Technology Stack

Component	Technology/Tool	Purpose
Programming Language	Python	Core development, AI integration.
Robotic Process Automation (RPA)	Automation Anywhere/any other RPA tool	Download documents from the portal, Email automation.
Document Automation System (DAS)	Mayan EDMS	Document storage, metadata extraction, and processing.
AI Large Language Model, Generative AI	Mistral 7B	Document extraction, Email extraction, Loan document analysis, risk assessment, natural language understanding.
	Retrieval Augmented Generation (RAG) System	Stores bank policies and loan criteria, used for evaluation by AI Agent.

5. Design Flow

The Agentic AI Underwriter follows the below execution steps:

1. **Data Collection**
   - The Loan applicant uploads the required documents (W2, pay stubs, bank statements, address and ID proof, photograph etc.) to the loan application portal.
   - The sales representative emails the bank loan department with key loan details, including purchase price, taxes, down payment, and the applicants income.
2. **Document Extraction**
   - A software robot (bot) downloads the documents from the loan application portal
   - The Documents are sent to **Mayan EDM**S a Document Automation System (DAS) for data extraction (e.g. applicant's name, address, income, etc.) using **Mistral 7B** – a large language model (LLM).

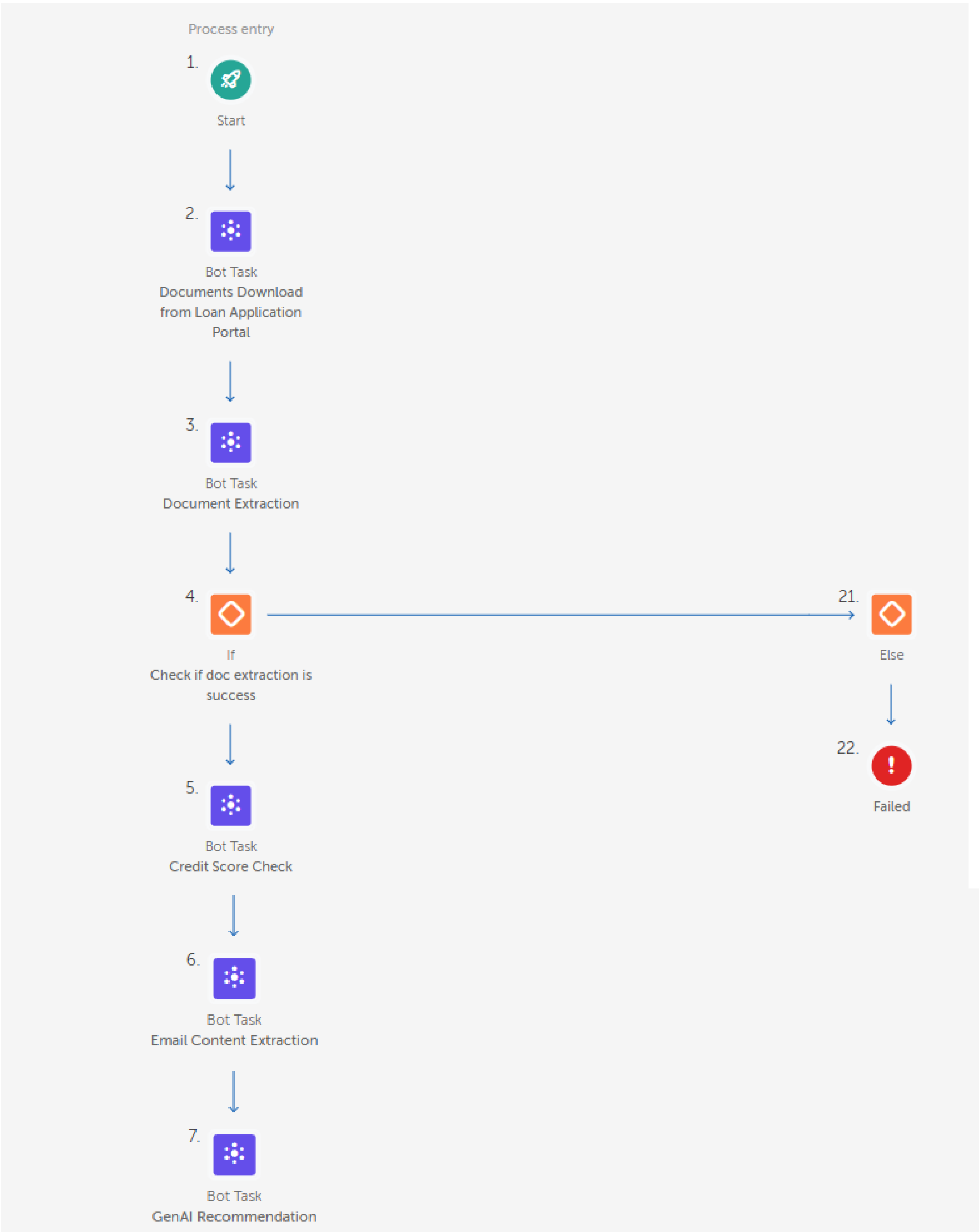
3. **Credit Score**
    - The bot pulls the applicant credit score from credit bureaus via APIs (e.g. Experian API, Equifax API) using **Python.**
4. **Email extraction**
    - A Mistral 7B **Generative AI powered Email Extractor** processes the sales representative's email.
    - It extracts the relevant loan details (loan amount, income, down payment, taxes, etc.).
5. **Loan Recommendation**:
    - The extracted data (loan amount, income, credit score, etc.) is sent to the **Mistral 7B Loan Recommendation AI Agent.**
    - The AI Agent evaluates the application against bank policies, and loan criteria stored in a Retrieval Augmented Generation (RAG) system using **Mistral 7B.**
    - Generates a loan recommendation based on compliance and risk assessment.
6. **Decisioning**
    - **Approval Process**
        i. If the AI Agent recommends approval, it highlights risk and proposed loan terms.
        ii. A human-in-loop task is triggered, allowing a loan underwriter to review and validate the recommendation.
        iii. The underwriter submits the final decision, initiating further action.

    - **Denial Process**
        i. If the AI Agent detects any non-compliance policy, it recommends the denial.
        ii. Similar to approval task, a human review task is triggered to validate the denial decision.
7. **Communication**
    - The **Agentic AI System** updates the bank's **Loan Management System** with the final loan status using **API**.
    - The bot sends the automated emails to the respective stakeholders (applicant, sales rep, loan department) regarding the approval/denial.

6. System Flow Summary

1. **Data Collection** → Loan application submission.
2. **Document Extraction** → OCR and structured data extraction.
3. **Credit Score Retrieval** → API calls to credit bureaus using Python.
4. **Email Extraction** → Mistral 7B Generative AI for email processing.
5. **Loan Recommendation** → AI-driven risk assessment & recommendation using Mistral 7B and RAG.
6. **Decisioning** → Human review triggered for approval/denial.
7. **Communication** → Automated notifications & system updates using API.

7. Flow Diagram



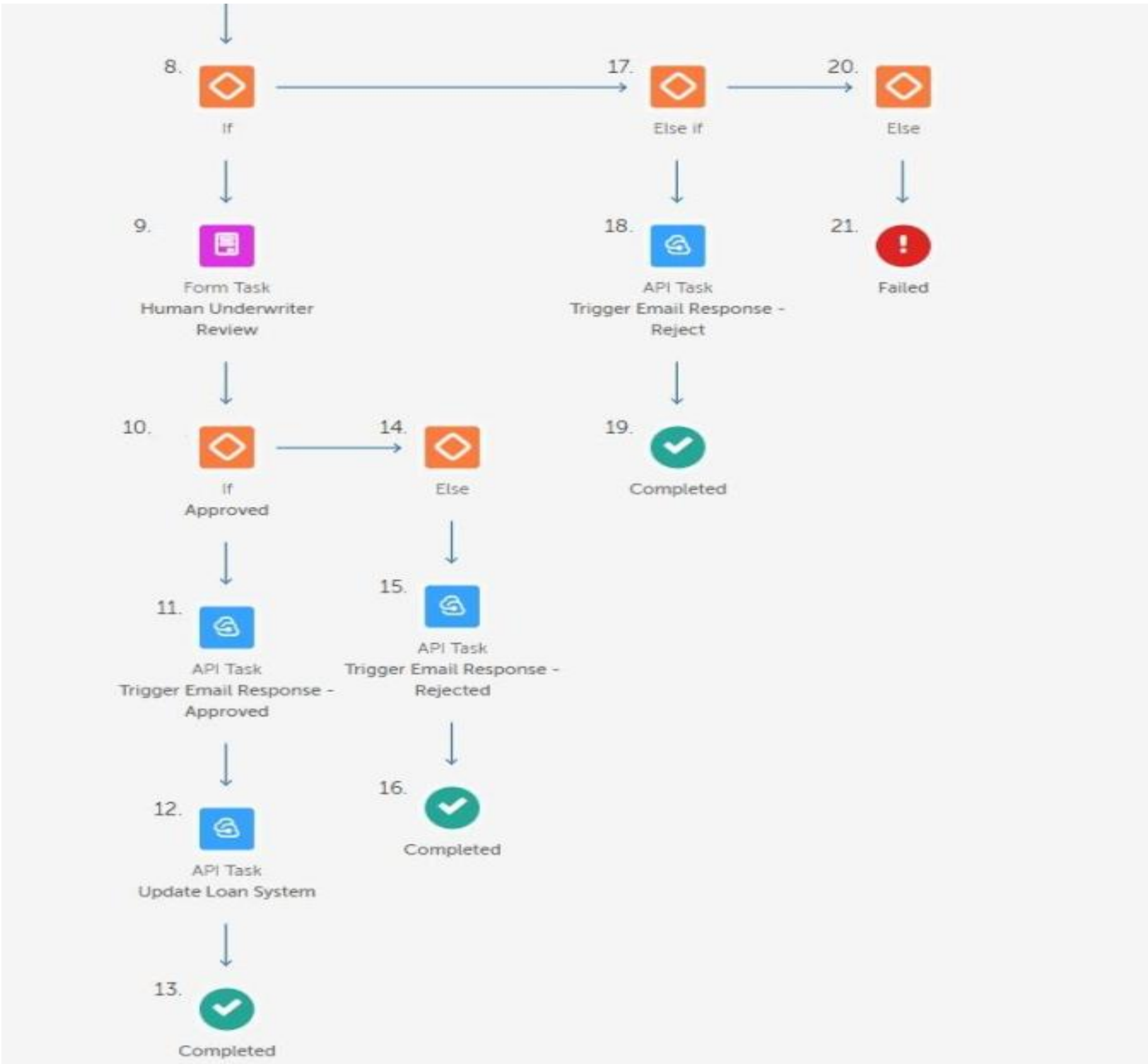


Figure 2. Process Flow Diagram

8. Technical Design

This section explains the implementation approach for the Agentic AI Underwriter.

Step 1: Download documents from Loan Application portal using Python API Integration

1. Setup and Configuration:
    a. Define the base_url for the API.
    b. Set up the API_KEY for authentication.
    c. Create a directory for storing downloaded documents.
2. Initialize API Session:

a. Create a session object with headers containing the Authorization token.

3. Fetch Loan Applications:
   a. Send a GET request to the /applications endpoint.
   b. Parse the response to get a list of application IDs.
4. Fetch Documents for Each Application:
   a. For each application ID:
      i. Send a GET request to /applications/{application_id}/documents.
      ii. Parse the response to get a list of document metadata (e.g., document ID, type).
5. Download Documents:
   a. For each document in the list:
      i. Send a GET request to /documents/{document_id}/download.
      ii. Save the document to the local directory with a unique filename (e.g., including application ID, document type, and timestamp).
6. Send Documents to Mayan EDMS – Document Automation System (DAS)
   a. For each downloaded document:
      i. Open the file and prepare it for upload.
      ii. Send a POST request to Mayan EDMS via it's REST API to the upload endpoint with the document file and metadata (e.g., application ID, document type).
      iii. Log the response from the automation system.
7. Logging:
   a. Log all activities, including successful downloads and errors.
8. Create Document Index:
   a. Generate a CSV file containing metadata for all downloaded documents (e.g., application ID, document type, filename).
9. Error Handling:
   a. Handle network errors, authentication failures, and invalid responses gracefully.

Step 2: Documents Extraction using Mistral 7B Large Language Model (LLM)

Load the Mistral 7B LLM model	model = load_mistral_7b_model()
Preprocess the document for input (text extraction)	text = preprocess_document(document)
Send the document text to Mistral 7B model for data extraction	extracted_data = model.extract_information(text)
Extract specific data points from the model output	applicant_name = extracted_data["name"] applicant_address = extracted_data["address"] applicant_income = extracted_data["income"] applicant_id = extracted_data["id"] applicant_photo = extracted_data["photo"]
Return the extracted data	return { "name": applicant_name, "address": applicant_address, "income": applicant_income, "id": applicant_id, "photo": applicant_photo }
Load the pre-trained Mistral 7B model (from local storage or server)	model = load_model("Mistral_7B") return model
Step to extract text from document if it's in PDF, image, or other formats	if document is image: text = extract_text_from_image(document) else if document is PDF: text = extract_text_from_pdf(document) else: text = document.text
Clean and normalize the extracted text	clean_text = clean_text(text) return clean_text

Remove unnecessary characters and normalize the text	cleaned_text = remove_special_characters(text) cleaned_text = correct_ocr_errors(cleaned_text) return cleaned_text
Use an OCR tool like Tesseract to extract text from an image	return ocr_tool.extract_text(image)
Extract text from the PDF using a PDF parser	return pdf_parser.extract_text(pdf)

Step 3: Credit Score Pull from Credit Bureaus API using Python

Get access token	import requests
	import json
	from datetime import datetime
	
	def authenticate_equifax(api_key, client_id):
	  """
	  Authenticate with Equifax API and retrieve an access token.
	  """
	  print("Authenticating with Equifax API...")
	  auth_url = "https://api.equifax.com/v1/auth"
	  headers = {
	    'client_id': client_id,
	    'api_key': api_key
	  }
	
	return "mock_equifax_token"

Using the access token, get the Equifax score	def get_equifax_credit_score(access_token, applicant_info):
	  """
	  Retrieve credit score from Equifax API.
	  """
	  print("Retrieving credit score from Equifax API...")
	  endpoint = "https://api.equifax.com/v1/credit-score"
	  headers = {
	    'Authorization': f'Bearer {access_token}',
	    'Content-Type': 'application/json'
	  }
	
	  payload = {
	    'consumerPII': {
	      'name': {
	        'firstName': applicant_info['first_name'],
	        'lastName': applicant_info['last_name']
	      },
	      'ssn': applicant_info['ssn'],
	      'dateOfBirth': applicant_info['dob'],
	      'currentAddress': {
	        'street': applicant_info['address']['street'],
	        'city': applicant_info['address']['city'],
	        'state': applicant_info['address']['state'],
	        'zip': applicant_info['address']['zip'] } } }

Equifax API credentials. Authenticate and retrieve credit score.	def main(): api_key = "EQUIFAX_API_KEY" client_id = " EQUIFAX_CLIENT_ID" access_token = authenticate_equifax(api_key, client_id) credit_score = get_equifax_credit_score(access_token, applicant_info)
Output	print("\\ Equifax Credit Score Result:") print(json.dumps(credit_score, indent=2))

Example of applicant information	applicant_info = {
	'first_name': 'Mohammad Asif',
	'last_name': 'Ali',
	'ssn': '333-44-6666', # In real implementation: Handle SSN securely!
	'dob': '1990-01-01',
	'address': {
	'street': '620 Liberty Ave,
	'city': 'Pittsburgh',
	'state': 'PA',
	'zip': '16046'
	}
	}

Example of output/response	return {
	'bureau': 'equifax',
	'credit_score': 750,
	'score_type': 'FICO',
	'score_date': ‘2025-01-28T15:20:45’,
	'status': 'success'
	}

Step 4: Email Extraction using Mistral 7B Generative AI Extractor (Hugging Face Library)

Install Hugging Face Transformers Library using Bash command	pip install transformers torch
Import Statement	from transformers import AutoModelForCausalLM, AutoTokenizer import torch
Load Mistral 7B model and token from Hugging Face	def extract_loan_details_from_email(email_text): # model_name = "mistralai/Mistral-7B" tokenizer= AutoTokenizer.from_pretrained(model_name) model= AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16, device_map="auto")
Define structured prompt	prompt = f""" Extract key loan details from the following email: - Loan Amount - Income - Down Payment - Taxes - Other relevant financial information Email Content: {email_text} """

Input Tokenization	input_tokens= tokenizer(prompt, return_tensors="pt").to("cuda")
Generate response using Mistral 7B	output_tokens = model.generate(**input_tokens, max_length=512)
Decode response	extracted_data = tokenizer.decode(output_tokens[0], skip_special_tokens=True)
Return statement	return extracted_data
Example	email_text = """ Hello Loan Dept, The applicant has an annual income of $115,000 and is applying for a $70,000 loan. The down payment is $10,000, and property taxes are estimated at $2,000 per year. Regards, Sales Rep """
Output	loan_details = extract_loan_details_from_email(email_text) print("Extracted Loan Details:\n", loan_details) yaml Extracted Loan Details: Loan Amount: $70,000 Income: $115,000 Down Payment: $10,000 Taxes: $2,000 per year Other relevant financial information: None mentioned

Step 5: Loan Recommendation integrated with RAG

Import Statement	from transformers import AutoModelForCausalLM, AutoTokenizer import torch
Load Mistral 7B Loan Recommendation AI Agent	model_name = "mistralai/Mistral-7B" tokenizer= AutoTokenizer.from_pretrained(model_name) model= AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16,device_map="auto") def generate_loan_recommendation(loan_data, bank_policies): """ Evaluates loan application against bank policies and generates a recommendation. Args: loan_data (dict): Extracted loan details (loan amount, income, credit score, etc.). bank_policies (str): Retrieved bank policies from the RAG system. Returns: str: Loan recommendation response. """
Design structured prompt for AI evaluation	prompt = f""" You are an AI Loan Underwriter evaluating a loan application.

	Loan Application Details: - Loan Amount: ${loan_data['loan_amount']} -Income: ${loan_data['income']} -Credit Score: {loan_data['credit_score']} -Down Payment: ${loan_data['down_payment']} -Taxes: ${loan_data['taxes']} Bank Policies and Loan Criteria: {bank_policies} Based on the above details, assess the compliance and risk of this loan application. Provide a structured recommendation including: - Approval or Denial decision. - Reasoning based on risk assessment. - Proposed loan terms (if approved). """
Input Tokenization	input_tokens= tokenizer(prompt, return_tensors="pt").to("cuda")
Generate AI response	output_tokens= model.generate(**input_tokens, max_length=512)
Decode response	recommendation = tokenizer.decode(output_tokens[0], skip_special_tokens=True)
Return Statement	return recommendation
Example	loan_data = {"loan_amount": 70000, "income":115000, "credit_score": 750, "down_payment": 10000, "taxes": 2000 }
Retrieved bank policies from RAG system	bank_policies = """ Minimum Credit Score: 650 Maximum Debt-to-Income Ratio: 35% Loan-to-Value (LTV) Ratio should not exceed 75%. """
Generate loan recommendation	recommendation= generate_loan_recommendation(loan_data, bank_policies)
Print AI Loan Recommendation	print("Loan Recommendation:\n", recommendation)

Step 6: Decisioning Approval/Denial

Import Statements	from mistralai.client import MistralClient from airflow import DAG from airflow.operators.python import PythonOperator from datetime import datetime

AI Model	api_key=
	os.getenv("MISTRAL_API_KEY", "actual_api_key") # Fetch API Key from environment variables
	ai_client = MistralClient(api_key=api_key) # Initialize AI

	model with dynamic API key
	
	def create_human_review_task(application_data, ai_decision, risk_assessment, proposed_terms=None):
	
	  """Creates a human-in-the-loop task for review."""
	  task_data = {
	    "application_data": application_data,
	    "ai_decision": ai_decision,
	    "risk_assessment": risk_assessment,
	    "proposed_terms": proposed_terms,
	  }
	  print(f"Human review task created with data: {task_data}")
	
	class LoanUnderwriterAgent:
	  def __init__(self, application_data):
	    self.application_data = application_data
	    self.ai_decision = None
	    self.risk_assessment = None
	    self.proposed_terms = None
	
	  def evaluate_application(self):
	    """Use Mistral 7B to analyze loan application and provide decisioning."""
	    prompt = f"""
	    Evaluate the following loan application:
	    {self.application_data}
	    Provide a decision (Approve/Denial), risk assessment, and loan terms if approved.
	    """
	    response = ai_client.generate(model="mistral-7b", prompt=prompt, max_tokens=200)
	    self.ai_decision = response.get("decision")
	    self.risk_assessment = response.get("risk_assessment", "No risk assessment available")
	    self.proposed_terms = response.get("proposed_terms", "No terms available")
	    return response
	
	  def process_decision(self):
	    """Trigger human review based on AI decision."""
	    create_human_review_task(self.application_data, self.ai_decision, self.risk_assessment, self.proposed_terms if self.ai_decision == "Approve" else None)
	
	  def finalize_decision(self, human_decision):
	    """Finalizes the decision based on human review."""
	    if human_decision not in ["Approve", "Deny"]:
	      raise ValueError("Invalid final decision")
	    print(f"Final decision submitted: {human_decision}")
	    return human_decision

Define Airflow DAG to integrate the loan	with DAG("loan_underwriting", start_date=datetime(2025, 2, 1),

<table>
<tr><td>underwriting workflow into Apache Airflow for Workflow Automation.</td><td>

```
schedule_interval=None, catchup=False) as dag:
    # Get application data dynamically (from arguments)
    application = os.getenv("APPLICATION_DATA")    # Get from environment variable or configuration

    if not application:
        raise ValueError("Application data not provided")

    underwriter = LoanUnderwriterAgent(application)

    evaluate_task = PythonOperator(
        task_id="evaluate_application",
        python_callable=underwriter.evaluate_application
    )

    process_task = PythonOperator(
        task_id="process_decision",
        python_callable=underwriter.process_decision
    )

    evaluate_task >> process_task
```

</td></tr>
<tr><td>Example to call the code – Approved case</td><td>

```
if __name__ == "__main__":
        application_data_approve = {
        "applicant_name": "Jo Smith",
        "credit_score": 690,
        "income": 115000,
        "loan_amount": 80000,
        "policy_compliance": True
    }

    underwriter_approve=
LoanUnderwriterAgent(application_data_approve)
    decision_approve=
underwriter_approve.evaluate_application()
    print(f"AI Decision: {decision_approve}")
    underwriter_approve.process_decision()
    final_decision_approve="Approve”
underwriter_approve.finalize_decision(final_decision_approve)
```

</td></tr>
<tr><td>Example to call the code – Denied case</td><td>

```
application_data_deny = {
        "applicant_name": "Adam Stone",
        "credit_score": 550,
        "income": 35000,
        "loan_amount": 550000,
        "policy_compliance": False
    }

    underwriter_deny=
LoanUnderwriterAgent(application_data_deny)
    decision_deny = underwriter_deny.evaluate_application()
    print(f"AI Decision: {decision_deny}")
    underwriter_deny.process_decision()
 final_decision_deny = "Deny"
```

</td></tr>
</table>

	underwriter_deny.finalize_decision(final_decision_deny)
Expected output – Approved case	AI Decision: {'decision': 'Approve', 'risk_assessment': 'Low risk due to good credit score and policy compliance', 'proposed_terms': 'Interest rate 4.5%, 15-year term'} Human review task created with data: {'application_data': {'applicant_name': 'Jo Smith', 'credit_score': 690, 'income': 115000, 'loan_amount': 80000, 'policy_compliance': True}, 'ai_decision': 'Approve', 'risk_assessment': 'Low risk due to good credit score and policy compliance', 'proposed_terms': 'Interest rate 4.5%, 15-year term'} Final decision submitted: Approve
Expected output – Denied case	AI Decision: {'decision': 'Deny', 'risk_assessment': 'High risk due to low credit score, insufficient income, high loan amount, and non-compliance with policy', 'proposed_terms': 'N/A'} Human review task created with data: {'application_data': {'applicant_name': 'Adam Stone', 'credit_score': 550, 'income': 35000, 'loan_amount': 550000, 'policy_compliance': False}, 'ai_decision': 'Deny', 'risk_assessment': 'High risk due to low credit score, insufficient income, high loan amount, and non-compliance with policy', 'proposed_terms': 'N/A'} Final decision submitted: Deny

Step 7: Communication

- *Update Loan status to Loan Management System using API*

Header	import requests def update_loan_status(loan_id, final_decision): api_url= "https://api.loan-management-system.com/update_status" api_key= "actual_api_key" # Use environment variable for security headers = { "Authorization": f"Bearer {api_key}", "Content-Type": "application/json" }

Payload	payload =
	{ "loan_id": loan_id,
	"status": final_decision }
	
	response = requests.post(api_url, json=payload, headers=headers)
	
	if response.status_code == 200:
	print(f"Loan status updated successfully for Loan ID: {loan_id}")
	else:

	print(f"Failed to update loan status for Loan ID: {loan_id}. Error: {response.text}")
Example	update_loan_status("33456745", "Approved") update_loan_status("52256745", "Denied")

- *Python code to trigger Automation Anywhere for email notification*

Sends an email using an Automation Anywhere bot.	import requests def send_email_via_bot(subject, recipient_email, message_body): bot_url= "https://bot.api.automationanywhere.com/send_email" bot_api_key = "actual_bot_api_key" # Use environment variable for security

Header	headers = {
	"Authorization": f"Bearer {bot_api_key}",
	"Content-Type": "application/json"
	}

Payload	payload = {
	"subject": subject,
	"recipient_email": recipient_email,
	"message_body": message_body }
	
	response = requests.post(bot_url, json=payload, headers=headers)
	
	if response.status_code == 200:
	print(f"Email successfully sent to {recipient_email}")
	else:
	print(f"Failed to send email to {recipient_email}. Error: {response.text}")

Usage Example	def notify_stakeholders(loan_decision, applicant_email, sales_rep_email, loan_dept_email):
	
	# Dynamic email messages
	subject = f"Loan Application Decision: {loan_decision}"
	message_body = f"Dear Stakeholder, the loan application has been {loan_decision}. Please review the details."
	#Send to applicant
	send_email_via_bot(subject, applicant_email, message_body)
	
	# Send to sales rep send_email_via_bot(subject, sales_rep_email, message_body)
	
	# Send to loan department send_email_via_bot(subject, loan_dept_email, message_body)

Example	notify_stakeholders("Approved", "applicant@ymail.com","salesrep@ymail.com", "loan_dept@ymail.com") notify_stakeholders("Denied", "applicant@ymail.com","salesrep@ymail.com", "loan_dept@ymail.com")

- *Integrate Loan Management System and Email Notification*

Update Loan statis in LMS	def finalize_and_notify(loan_id, final_decision, applicant_email, sales_rep_email, loan_dept_email): update_loan_status(loan_id, final_decision)
Email notification to stakeholders	notify_stakeholders(final_decision, applicant_email, sales_rep_email, loan_dept_email)
Example	finalize_and_notify("443453543","Approved", "applicant@ymail.com","salesrep@ymail.com", "loan_dept@ymail.com") finalize_and_notify("653453543","Denied", "applicant@ymail.com","salesrep@ymail.com", "loan_dept@ymail.com")

9. Conclusion & Future Work

The application of Agentic AI in the loan underwriting process signifies a major shift within the financial sector, providing improvements in efficiency, precision, and scalability. By utilizing large language models (LLMs), retrieval-augmented generation (RAG), and robotic process automation (RPA), this methodology automates repetitive functions, optimizes decision-making, and lowers operational expenses.

Through the adoption of Agentic AI, financial institutions can establish more agile, transparent, and scalable underwriting system, ultimately facilitating greater loan accessibility and promoting financial inclusion.

However, challenges related to AI explainability, regulatory compliance, and model bias persists. Future research should focus on:

1. Enhancing AI interpretability to increase transparency in loan decisions.
2. Developing federated learning models for secure and unbiased credit risk analysis.
3. Expand the testing across different financial institutions to validate AI's impact further.

By addressing these areas, Agentic AI can revolutionize the underwriting industry, improving financial inclusion while maintaining ethical AI governance.

References

[1] https://www.microsoft.com/en-us/research/blog/autogen-v0-4-reimagining-the-foundation-of-agentic-ai-for-scale-extensibility-and-robustness/
[2] https://www.automationanywhere.com/rpa/agentic-ai
[3] https://www.uipath.com/ai/agentic-ai
[4] https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/
[5] https://spectrum.ieee.org/ai-agents
[6] https://www.salesforce.com/
[7] https://ieeexplore.ieee.org/document/10047188
[8] https://arxiv.org/html/2411.17598v1
[9] Book: “The Age of AI: And Our Human Future” by Henry A Kissinger
[10] Book: “Agentic AI: Harnessing the Power of Autonomous Systems in the Modern Workplace” by Taylor Royce
[11] Book: “The AI Advantage: How to Use AI to Underwrite Multifamily Development” by Tim Safransky
[12] Book: “The Digitally-Enabled Underwriter” by Ron Glozman.
[13] “Enhancing AI Systems with Agentic Workflows Patterns in Large Language Model”, IEEE Xplore, 2024.
[14] "Explainable AI for Underwriting: Balancing Performance and Compliance," IEEE Xplore, 2024.
[15] “Agentic AI: Autonomous Intelligence for Complex Goals – A Comprehensive Survey,” ResearchGate, 2025.