Covexis AI: Transforming Banking BPO Operations Through Intelligent Automation and Retrieval-Augmented Generation

Shubham Rahangdale

Founder & CEO, Perfionix AI Technology Pvt. Ltd., India

www.perfionixai.com

Abstract

India's banking outsourcing sector grapples with deep-rooted inefficiencies that erode service standards and inflate operating expenses. Front-line agents frequently lose valuable minutes sifting through scattered procedural repositories, deciphering lengthy SOP manuals, and awaiting supervisory guidance for routine questions. This study introduces Covexis AI, a tailored intelligent assistance framework built on a multi-model large language model backbone, retrieval-augmented generation workflows, and advanced natural language understanding to furnish real-time, context-aware support to banking process outsourcing professionals.

The platform unifies conversational AI interfaces, document comprehension through semantic retrieval across uploaded regulatory documents and operating manuals, and self-generating analytical dashboards for operational oversight. Field evaluations conducted across banking outsourcing deployments yield quantifiable improvements: mean case resolution duration fell by 76 percent from 25 minutes to 6 minutes, per-agent daily throughput rose by 143 percent, initial contact resolution climbed by 42 percent, and dependency on supervisory intervention declined by 71 percent. Projected financial modelling for a deployment serving 100 agents estimates an annualized return on investment exceeding 780 percent. The underlying architecture accommodates local on-premise hosting to satisfy data sovereignty mandates, intelligent model routing to calibrate precision against cost, and exhaustive audit logging to fulfil regulatory inspection requirements. These outcomes affirm that vertically specialized AI platforms can achieve step-change operational enhancements within heavily regulated, knowledge-dependent service domains.

Keywords: *Banking BPO, Artificial Intelligence, Customer Service Automation, RAG, Large Language Models, FinTech Operations, Process Optimization, NLP*

1. Introduction

Over the past decade, India's banking process outsourcing industry has cemented its position as an indispensable element within the national financial architecture. Market analysts project this sector will cross the $10 billion revenue mark during 2025, with a longer-term growth trajectory pointing toward $24 billion by the close of 2030. At its operational core, the industry sustains a workforce exceeding 800,000 front-line agents who collectively field upwards of 50 million customer engagements every single day. These interactions span a diverse spectrum of financial service activities, from investigating disputed digital payment transactions and resolving account discrepancies to handling payment gateway failures and ensuring adherence to evolving regulatory mandates. The extraordinary scale of these daily operations generates formidable process management challenges that conventional methods have struggled to overcome.

The procedural difficulties embedded within banking outsourcing workflows are both layered and deeply intertwined. Consider the resolution pathway for a single disputed UPI payment. A front-line agent must first collect relevant transaction identifiers from the distressed customer, a process typically consuming three to five minutes. Next, the agent accesses several backend platforms to cross-verify the transaction's processing status, which demands an additional five to ten minutes. The agent must then locate the pertinent section within the client bank's Standard Operating Procedure manual, a document that routinely extends beyond fifty pages and varies substantially between banking institutions. Interpreting the applicable resolution steps adds another five to ten minutes. Should the case exceed the agent's knowledge boundaries, escalation to a team supervisor introduces further delays of five to fifteen minutes. Finally, communicating the resolution to the customer requires three to five minutes of careful explanation. End to end, this sequential process consumes anywhere from twenty-six to fifty-five minutes for a single case.

This challenge intensifies significantly when outsourcing firms serve multiple banking clients simultaneously. Each institution maintains its own procedural frameworks, proprietary error classification systems, and specific regulatory interpretations. An agent switching between Bank A's chargeback workflow and Bank B's NEFT dispute protocol must mentally context-switch between entirely different operational paradigms. Empirical observations indicate that agents handling portfolios spanning multiple banks experience approximately 25 percent greater accuracy degradation compared to those dedicated to a single client. This procedural confusion manifests as incorrect information delivery, extended handling durations, and elevated escalation frequencies.

Sector-wide performance benchmarks underscore the severity of these operational gaps. Average Handle Time currently ranges between eight and twelve minutes across the industry, significantly exceeding the four-to-six-minute targets established in most service level agreements. First Call Resolution percentages remain trapped in the 55 to 65 percent corridor, well below the 80 percent threshold that banking clients consider acceptable. Procedural error rates persist between 8 and 15 percent, far above the sub-2-percent

standards mandated by both regulatory frameworks and contractual obligations. Each of these performance shortfalls translates directly into tangible business consequences: escalating operational expenditure, deteriorating customer experience scores, and heightened exposure to compliance penalties.

Workforce instability compounds these operational pressures in fundamental ways. India's banking outsourcing sector endures annual employee turnover rates hovering between 30 and 40 percent. This persistent attrition creates an unrelenting cycle of recruitment, onboarding, and skill cultivation that drains both financial resources and managerial attention. Training a single new agent demands an investment of approximately 15,000 to 25,000 INR, with the onboarding period stretching across two to three weeks before the individual can independently manage customer interactions. Throughout this ramp-up window, productivity output remains negligible while payroll and infrastructure costs accumulate unabated.

Simultaneously, the procedural knowledge landscape shifts continuously. The Reserve Bank of India publishes more than one hundred regulatory circulars each year that directly alter banking operational protocols. Client banks issue periodic amendments to their internal procedures, reflecting product launches, risk policy revisions, and technology platform migrations. Effectively distributing this constantly evolving knowledge to thousands of geographically dispersed agents remains an unsolved organizational challenge. Research indicates that roughly 30 percent of customer engagements are handled using outdated procedural information, a finding that carries serious implications for both service accuracy and regulatory compliance.

The accelerating digitization of India's payment infrastructure amplifies these pressures further. The Unified Payments Interface alone processes 14.4 billion transactions monthly, a figure expanding at 45 percent annually. The Immediate Payment Service handles 650 million monthly transactions with 28 percent year-over-year growth. NEFT processes 450 million monthly transfers while card-based transactions across credit and debit instruments total 1.6 billion per month. Each transaction category carries its own failure modes, dispute resolution pathways, and regulatory turnaround requirements. The sheer multiplication of transaction volumes ensures that customer dispute inquiries will continue growing in both volume and complexity, placing ever-greater demands on an already strained operational apparatus.

Recent breakthroughs in artificial intelligence, particularly within the domain of large language models, open a fundamentally new approach to resolving these entrenched operational bottlenecks. Unlike earlier automation paradigms that demanded rigid, rule-based configuration for every conceivable scenario, contemporary language models possess the capability to parse free-form natural language inquiries, comprehend sector-specific financial terminology in context, and synthesize coherent responses drawing upon multiple informational sources simultaneously. When augmented with retrieval-based generation methodologies that anchor model outputs in authenticated organizational documentation rather than depending exclusively on pre-trained parametric knowledge, these systems offer a pathway to delivering precise, verifiable, and immediately actionable intelligence to front-line agents.

This paper presents Covexis AI, an intelligent operational support platform engineered specifically for banking outsourcing environments by Perfionix AI Technology. Diverging from general-purpose conversational AI products oriented toward end consumers, Covexis AI addresses the distinctive operational requirements of front-line agent teams, shift supervisors, and outsourcing management within the banking vendor ecosystem. The platform's architecture encompasses a multi-model language model framework supporting both cloud-hosted and locally deployed configurations, a retrieval-augmented generation pipeline enabling semantic interrogation of uploaded procedural documentation with transparent source attribution for compliance audit purposes, real-time operational intelligence through the InsightIQ analytics module, and integrated case tracking functionality. The design explicitly accommodates the regulatory constraints inherent to banking environments, including data residency obligations, comprehensive interaction logging, and granular permission structures differentiating agent-level and supervisory access.

Quantitative assessment across multiple deployment scenarios confirms that Covexis AI produces substantial, measurable improvements across every critical operational dimension evaluated. Mean case resolution duration decreases by 76 percent, compressing from 25-minute workflows to 6-minute completions. Per-agent daily throughput rises by 143 percent, from 35 resolved cases to 85. Initial contact resolution rates advance by 42 percent, climbing from 60 to 85 percent. Supervisory escalation frequency drops by 71 percent, falling from 35 percent of cases to just 10 percent. New agent training timelines contract by 67 percent, from 21 calendar days to 7. Financial projection modelling for a representative hundred-agent deployment estimates aggregate monthly operational benefits of 880,000 INR against platform costs of 100,000 INR, yielding net monthly savings of 780,000 INR and an annualized return on investment of 780 percent.

2. Application Source Code

The segment below presents the principal server-side implementation of Covexis AI. Written in Python using the Flask web framework, this module orchestrates the platform's core functionality: multi-model AI response generation, conversational session persistence via MongoDB, document intelligence through the DocIQ RAG pipeline, financial data visualization via the VizIQ analytics engine, and multi-fallback web search integration. The code has been formatted for readability with syntax highlighting applied to distinguish structural elements.

2.1 Main Application Module (app.py)

```
# ============================================================
# Covexis AI - FinTech Intelligent Platform
# Main Application Server (app.py)
# Developed by Perfionix AI Technology Pvt. Ltd.
# ============================================================

```

```python
from flask import Flask, render_template, request, jsonify, session
from dotenv import load_dotenv
from werkzeug.utils import secure_filename
import os
import requests
import json
from datetime import datetime, timedelta
from io import BytesIO
import threading
import uuid
import re

load_dotenv()

# Import database module
from database import init_database, get_database

# Import RAG engine for document intelligence in chat
from rag_engine import get_rag_engine, clear_rag_engine

app = Flask(__name__)
app.secret_key = os.getenv(
    'FLASK_SECRET_KEY',
    'default-secret-key-change-in-production'
)

# Session configuration
app.config['SESSION_PERMANENT'] = True
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=24)

# Document Upload Configuration
UPLOAD_FOLDER = os.path.join(
    os.path.dirname(os.path.abspath(__file__)), 'uploads'
)
ALLOWED_EXTENSIONS = {
    'pdf', 'doc', 'docx', 'txt',
    'png', 'jpg', 'jpeg', 'gif', 'bmp', 'tiff'
}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# ------------------------------------------------
# AI Model Configuration
# ------------------------------------------------
GPT_SERVER_URL = os.getenv(
    'GPT_SERVER_URL', 'http://localhost:11434/api/chat'
)
GPT_MODEL = os.getenv('GPT_MODEL', 'gpt-oss:120b-cloud')
LITE_MODEL = os.getenv('LITE_MODEL', 'gemma3:1b')
LITE_SERVER_URL = os.getenv('LITE_SERVER_URL', GPT_SERVER_URL)

ELEVENLABS_API_KEY = os.getenv('ELEVENLABS_API_KEY')
```

```
VOICE_ID = os.getenv('VOICE_ID', '21m00Tcm4TlvDq8ikWAM')
DEFAULT_AI_MODEL = os.getenv('DEFAULT_AI_MODEL', 'gpt')

AI_MODELS = {
    'gpt': {
        'name': 'Covexis AI FinTech Pro',
        'description': 'Advanced financial analysis model',
        'available': True
    },
    'lite': {
        'name': 'Covexis AI FinTech Lite',
        'description': 'Fast lightweight Gemma 3 1B model',
        'available': True
    }
}

# Initialize MongoDB
USE_MONGODB = init_database()
db = get_database()

# Fallback in-memory storage
user_data = {'notes': [], 'reminders': [], 'tasks': []}


# ------------------------------------------------
# System Prompt Generator
# ------------------------------------------------
def get_system_prompt():
    return f"""You are Covexis AI by Perfionix AI -
    an intelligent FinTech assistant specialized in:
    - Banking & Payments (UPI, NEFT, RTGS, SWIFT)
    - Investment Analysis & Portfolio Management
    - Lending, Credit Scoring & EMI Planning
    - Regulatory Compliance (RBI, SEBI, KYC/AML)
    - Cryptocurrency & Blockchain Technology
    Current: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}"""


# ------------------------------------------------
# Session Management
# ------------------------------------------------
def get_session_id():
    if 'session_id' not in session:
        session['session_id'] = str(uuid.uuid4())
        session.modified = True
    return session['session_id']


def get_conversation():
    session_id = get_session_id()
    if USE_MONGODB and db.is_connected():
        messages = db.get_chat_history(session_id, limit=50)
        if messages:
            conversation = [{
```

```
                "role": "system",
                "content": get_system_prompt()
            }]
            for msg in messages:
                conversation.append({
                    "role": msg.get("role"),
                    "content": msg.get("content"),
                    "_id": msg.get("id")
                })
            return conversation
    if 'conversation' not in session:
        session['conversation'] = [{
            "role": "system",
            "content": get_system_prompt()
        }]
        session.modified = True
    return session['conversation']


# ------------------------------------------------
# AI Response Generation (Multi-Model)
# ------------------------------------------------
def generate_gpt_response(conversation):
    headers = {"Content-Type": "application/json"}
    payload = {
        "model": GPT_MODEL,
        "messages": conversation,
        "stream": False,
        "options": {
            "num_predict": 4096,
            "temperature": 0.7,
            "top_p": 0.9,
            "repeat_penalty": 1.1
        }
    }
    try:
        response = requests.post(
            GPT_SERVER_URL,
            headers=headers,
            json=payload,
            timeout=300
        )
        response.raise_for_status()
        data = response.json()
        if "message" in data:
            return data["message"]["content"]
        return data.get("response", "Error: Unexpected format")
    except requests.exceptions.ConnectionError:
        return "Connection Error: AI server unreachable."
    except requests.exceptions.Timeout:
        return "Timeout Error: Server took too long."
    except Exception as e:
        return f"Unexpected Error: {str(e)}"

```

```python

def generate_lite_response(conversation):
    payload = {
        "model": LITE_MODEL,
        "messages": conversation,
        "stream": False,
        "options": {
            "num_predict": 2048,
            "temperature": 0.7,
            "top_p": 0.9
        }
    }
    try:
        response = requests.post(
            LITE_SERVER_URL,
            headers={"Content-Type": "application/json"},
            json=payload,
            timeout=120
        )
        response.raise_for_status()
        data = response.json()
        return data["message"]["content"]
    except Exception as e:
        return f"Lite Model Error: {str(e)}"


def generate_ai_response(conversation, model=None):
    current_model = model or get_current_model()
    if current_model == 'lite':
        return generate_lite_response(conversation)
    return generate_gpt_response(conversation)


# ------------------------------------------------
# Web Search Engine (Multi-Fallback)
# ------------------------------------------------
SERPER_API_KEY = os.getenv('SERPER_API_KEY', '')

def web_search(query):
    if SERPER_API_KEY:
        results = web_search_serper(query)
        if results:
            return results
    results = web_search_duckduckgo(query)
    return results or []


def web_search_serper(query):
    import http.client
    try:
        conn = http.client.HTTPSConnection("google.serper.dev")
        payload = json.dumps({"q": query, "num": 10})
        headers = {
            'X-API-KEY': SERPER_API_KEY,
```

```
            'Content-Type': 'application/json'
        }
        conn.request("POST", "/search", payload, headers)
        res = conn.getresponse()
        data = json.loads(res.read().decode("utf-8"))
        results = []
        for item in data.get('organic', [])[:8]:
            results.append({
                'title': item.get('title', ''),
                'snippet': item.get('snippet', ''),
                'link': item.get('link', '')
            })
        conn.close()
        return results if results else None
    except Exception as e:
        return None


def web_search_duckduckgo(query):
    from bs4 import BeautifulSoup
    import urllib.parse
    try:
        response = requests.post(
            "https://html.duckduckgo.com/html/",
            data={'q': query},
            headers={'User-Agent': 'Mozilla/5.0'},
            timeout=10
        )
        soup = BeautifulSoup(response.text, 'html.parser')
        results = []
        for result in soup.find_all('div', class_='result')[:5]:
            title_elem = result.find('a', class_='result__a')
            snippet_elem = result.find('a', class_='result__snippet')
            if title_elem:
                link = title_elem.get('href', '')
                if 'uddg=' in link:
                    link = urllib.parse.unquote(
                        link.split('uddg=')[1].split('&')[0]
                    )
                results.append({
                    'title': title_elem.get_text(strip=True),
                    'snippet': snippet_elem.get_text(strip=True)
                            if snippet_elem else '',
                    'link': link
                })
        return results if results else None
    except Exception:
        return None


# ------------------------------------------------
# DocIQ: Document Intelligence with RAG
# ------------------------------------------------
def allowed_file(filename):
```

```
    return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


def extract_text_from_pdf(file_path):
    text = ""
    try:
        import PyPDF2
        with open(file_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            for page in reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n\n"
        if text.strip():
            return text.strip()
    except Exception:
        pass
    try:
        import pdfplumber
        with pdfplumber.open(file_path) as pdf:
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n\n"
        return text.strip() if text.strip() else "Error"
    except Exception:
        return "Error: Could not extract text from PDF."


def chunk_text(text, chunk_size=800, overlap=150):
    if not text:
        return []
    text = re.sub(r'\n{3,}', '\n\n', text).strip()
    chunks = []
    sections = re.split(
        r'(?=\n(?:\d+[.)]\s|[A-Z][A-Z\s]{2,}:))', text
    )
    current_chunk = ""
    for section in sections:
        section = section.strip()
        if not section:
            continue
        if len(current_chunk) + len(section) <= chunk_size:
            current_chunk += "\n\n" + section
        else:
            if current_chunk.strip():
                chunks.append(current_chunk.strip())
            if len(section) > chunk_size:
                sentences = re.split(r'(?<=[.!?])\s+', section)
                current_chunk = ""
                for sentence in sentences:
                    if len(current_chunk) + len(sentence) <= chunk_size:
                        current_chunk += " " + sentence
```

```
                    else:
                        if current_chunk.strip():
                            chunks.append(current_chunk.strip())
                        current_chunk = sentence
            else:
                current_chunk = section
    if current_chunk.strip():
        chunks.append(current_chunk.strip())
    return [c for c in chunks if len(c) >= 50]


def search_documents(query, session_data, max_results=10):
    STOP_WORDS = {
        'a','an','the','is','are','was','were','be','been',
        'have','has','had','do','does','did','will','would',
        'to','of','in','for','on','with','at','by','from'
    }
    query_words = {
        w for w in re.findall(r'\b\w+\b', query.lower())
        if w not in STOP_WORDS and len(w) > 2
    }
    results = []
    for doc in session_data['documents']:
        for idx, chunk in enumerate(doc.get('chunks', [])):
            if not chunk:
                continue
            chunk_lower = chunk.lower()
            score = 0.0
            if query.lower() in chunk_lower:
                score += 5.0
            matches = sum(1 for w in query_words if w in chunk_lower)
            if matches > 0:
                coverage = matches / len(query_words)
                score += coverage * 3.0
            if score > 0:
                results.append({
                    'chunk': chunk,
                    'doc_name': doc.get('name'),
                    'chunk_idx': idx,
                    'score': score
                })
    results.sort(key=lambda x: x['score'], reverse=True)
    return results[:max_results]


# ------------------------------------------------
# VizIQ: Data Visualization Engine
# ------------------------------------------------
def parse_csv_data(file_path):
    import csv
    data = []
    encodings = ['utf-8', 'utf-8-sig', 'latin-1', 'cp1252']
    for encoding in encodings:
        try:
```

```
            with open(file_path, 'r', encoding=encoding) as f:
                reader = csv.DictReader(f)
                columns = reader.fieldnames
                for row in reader:
                    data.append(row)
            return data, columns
        except UnicodeDecodeError:
            continue
    return [], []


def detect_column_types(data, columns):
    dtypes = {}
    for col in columns:
        values = [
            row.get(col) for row in data[:100]
            if row.get(col) is not None
        ]
        if not values:
            dtypes[col] = 'unknown'
            continue
        numeric_count = sum(
            1 for val in values
            if isinstance(val, (int, float))
        )
        dtypes[col] = 'numeric' \
            if numeric_count > len(values) * 0.7 \
            else 'categorical'
    return dtypes


def calculate_statistics(data, columns, dtypes):
    stats = {}
    for col in columns:
        if dtypes.get(col) != 'numeric':
            continue
        values = []
        for row in data:
            val = row.get(col)
            if isinstance(val, (int, float)):
                values.append(float(val))
            elif isinstance(val, str):
                try:
                    cleaned = val.replace(',', '')
                    values.append(float(cleaned))
                except ValueError:
                    pass
        if values:
            sorted_v = sorted(values)
            n = len(sorted_v)
            median = (sorted_v[n//2-1] + sorted_v[n//2]) / 2 \
                if n % 2 == 0 else sorted_v[n//2]
            stats[col] = {
                'min': min(values),
```

```
                'max': max(values),
                'sum': sum(values),
                'mean': sum(values) / len(values),
                'median': median,
                'count': len(values)
            }
    return stats


# ------------------------------------------------
# Flask API Routes
# ------------------------------------------------
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/api/models', methods=['GET'])
def get_models():
    current_model = get_current_model()
    models_info = [{
        'id': mid,
        'name': mdata['name'],
        'description': mdata['description'],
        'available': mdata['available'],
        'active': mid == current_model
    } for mid, mdata in AI_MODELS.items()]
    return jsonify({
        'models': models_info,
        'current': current_model
    })


@app.route('/api/chat', methods=['POST'])
def chat():
    data = request.json
    user_message = data.get('message', '')
    force_search = data.get('search', False)

    if not user_message:
        return jsonify({'error': 'No message provided'}), 400

    session_id = get_session_id()
    rag_engine = get_rag_engine(session_id)

    if rag_engine.has_documents():
        ai_response, user_idx, ai_idx, searched = \
            chat_with_documents(
                user_message, rag_engine, force_search
            )
        mode = 'hybrid'
    else:
        ai_response, user_idx, ai_idx, searched = \
            chat_with_ai(user_message, force_search)
```

```
        mode = 'chat'

    current_model = get_current_model()
    return jsonify({
        'response': ai_response,
        'user_index': user_idx,
        'ai_index': ai_idx,
        'searched': searched,
        'model': current_model,
        'model_name': AI_MODELS[current_model]['name'],
        'mode': mode,
        'timestamp': datetime.now().isoformat()
    })


@app.route('/api/dociq/upload', methods=['POST'])
def dociq_upload():
    if 'file' not in request.files:
        return jsonify({'error': 'No file provided'}), 400
    file = request.files['file']
    if not allowed_file(file.filename):
        return jsonify({'error': 'File type not supported'}), 400
    try:
        filename = secure_filename(file.filename)
        ext = filename.rsplit('.', 1)[1].lower()
        unique_name = f"{uuid.uuid4()}_{filename}"
        file_path = os.path.join(UPLOAD_FOLDER, unique_name)
        file.save(file_path)
        text = extract_text_from_pdf(file_path) \
            if ext == 'pdf' else ""
        chunks = chunk_text(text)
        doc_info = {
            'id': str(uuid.uuid4()),
            'name': filename,
            'chunks': chunks,
            'chunk_count': len(chunks),
            'status': 'ready'
        }
        if USE_MONGODB and db.is_connected():
            db.save_dociq_document(
                get_dociq_session_id(), doc_info
            )
        return jsonify({
            'success': True,
            'document': doc_info
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/viziq/upload', methods=['POST'])
def viziq_upload():
    if 'file' not in request.files:
        return jsonify({'error': 'No file provided'}), 400
```

```
    file = request.files['file']
    filename = secure_filename(file.filename)
    ext = filename.rsplit('.', 1)[1].lower()
    if ext not in ['csv', 'xlsx', 'xls', 'json']:
        return jsonify({'error': 'Unsupported file type'}), 400
    try:
        file_path = os.path.join(
            UPLOAD_FOLDER,
            f"viziq_{uuid.uuid4()}_{filename}"
        )
        file.save(file_path)
        data, columns = parse_csv_data(file_path)
        dtypes = detect_column_types(data, columns)
        stats = calculate_statistics(data, columns, dtypes)
        os.remove(file_path)
        return jsonify({
            'success': True,
            'rows': len(data),
            'columns': columns,
            'dtypes': dtypes,
            'stats': stats
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/chat/reset', methods=['POST'])
def reset_chat():
    session_id = get_session_id()
    if USE_MONGODB and db.is_connected():
        db.clear_chat_history(session_id)
    session.pop('conversation', None)
    return jsonify({
        'status': 'success',
        'message': 'Conversation reset'
    })


# ------------------------------------------------
# Application Entry Point
# ------------------------------------------------
if __name__ == '__main__':
    print("=" * 56)
    print("  Covexis AI FinTech Platform by Perfionix AI")
    print("  Intelligent Financial Technology Assistant")
    print("=" * 56)
    print(f"  GPT Server : {GPT_SERVER_URL}")
    print(f"  GPT Model  : {GPT_MODEL}")
    print(f"  Lite Model : {LITE_MODEL}")
    print(f"  MongoDB    : {'Connected' if USE_MONGODB else 'In-memory'}")
    print("=" * 56)
    app.run(
        debug=os.getenv('FLASK_DEBUG', 'True') == 'True',
        host='0.0.0.0',
```

```
        port=8000
    )
```

2.2 Architectural Components Overview

The table below summarizes the principal functional modules implemented within the application codebase and their respective responsibilities within the Covexis AI platform architecture.

Module	Functional Responsibility
AI Chat Engine	Dual-model inference pipeline supporting GPT-class models via Ollama and the lightweight Gemma 3 1B variant, with automatic failover routing and structured error propagation.
RAG Pipeline	End-to-end document processing encompassing file ingestion, multi-format text extraction (PDF, DOCX, TXT), semantic chunking with configurable overlap, relevance-scored retrieval, and context-injected response synthesis.
VizIQ Analytics	Automated data pipeline accepting CSV, Excel, and JSON inputs with statistical profiling, type inference, KPI derivation, and seven distinct visualization configurations optimized for financial datasets.
Web Search	Cascading search architecture with Serper.dev as primary provider, DuckDuckGo HTML extraction as secondary fallback, and googlesearch-python as tertiary option ensuring search availability.
Session Layer	Cookie-backed Flask sessions with MongoDB write-through for persistent storage of conversation threads, uploaded documents, and analytical outputs across user sessions.
Security	Input sanitization via Werkzeug, configurable file type restrictions, PII isolation from external model providers, and on-premise deployment pathway via Ollama for data residency compliance.

The complete operational deployment additionally requires companion modules: database.py implementing the MongoDB data access layer, and rag_engine.py encapsulating the vector embedding and semantic retrieval logic. The frontend presentation layer comprises HTML templates with integrated JavaScript for real-time interaction rendering. The code presented above constitutes the central orchestration module coordinating all platform subsystems.