Scientific Research Publishing

Open Journal of Applied Sciences, 2026, 16(6), 2161-2176
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Hierarchical Multi-Agent Systems for Automated Commercial Property Insurance Underwriting

Muhammad Imran Sajid

Al Rajhi Takaful, Riyadh, Saudi Arabia
Email: engrm.imran050@gmail.com

How to cite this paper: Sajid, M.I. (2026) Hierarchical Multi-Agent Systems for Automated Commercial Property Insurance Underwriting. *Open Journal of Applied Sciences*, **16**, 2161-2176.
https://doi.org/10.4236/ojapps.2026.166122

Received: May 12, 2026
Accepted: June 19, 2026
Published: June 22, 2026

Abstract

Commercial property insurance underwriting requires integrating evidence from heterogeneous document sources—financial statements, property inspection reports, legal certificates, and claims histories—a process that is time-consuming, labour-intensive, and prone to inconsistency. This paper presents a hierarchical multi-agent system that automates end-to-end commercial property underwriting using Large Language Models (LLMs), designed and evaluated in the context of the Kingdom of Saudi Arabia (KSA) regulatory framework. The proposed framework orchestrates nine specialised components through a directed LangGraph StateGraph workflow, processing three categories of input artifact: a structured JSON proposal form, an unstructured PDF/DOCX document bundle (financial, inspection, legal, and claims documents plus reference PDFs), and a domain-knowledge Excel template encoding Risk-Score Rating (RS-RR) criteria. A DocumentClassifier powered by GPT-4o-mini routes each document page into one of eight semantic buckets before four specialist GPT-4o agents analyse their respective domains in sequence. A ReferenceContextBuilder distils KSA underwriting guidelines, policy wordings, and exclusion clauses from reference PDFs, while a separate RiskEngineeringExtractor derives per-category scores across 26 fire-protection and physical-risk dimensions from the Excel template. A SynthesisAgent consolidates all specialist analyses and a DecisionAgent applies the organisation's risk methodology to produce a final risk rating and premium recommendation in Saudi Riyals (SAR). The system generates seven structured output artifacts including a KSA-aligned underwriting pack, a full LLM interaction log, and a scored risk-engineering assessment. Across five expert-designed commercial property proposals spanning the risk-class spectrum, the system completed each end-to-end underwriting in under three minutes, produced bit-identical outputs on repeated runs through a deterministic pre-

mium calculation, and assigned the risk class anticipated by the underwriters who designed each case. These results support the feasibility of bringing LLM-based multi-agent underwriting into regulated insurance markets; a larger expert-validated evaluation is identified as future work.

Keywords

Multi-Agent Systems, Large Language Models, Insurance Underwriting, Commercial Property Insurance, LangGraph, Document Classification, Risk Assessment, LangChain, Kingdom of Saudi Arabia

1. Introduction

Commercial property underwriting is, at its core, an exercise in reconciling evidence that speaks very different languages. An audited financial statement, a risk-engineer's site survey, a civil-defence certificate, and five years of claims history each demand a different kind of expertise to read correctly—and a human underwriter must somehow weigh all of them together before arriving at a single risk rating and premium figure. In practice, that synthesis rarely happens in less than a full working day, and complex cases can stretch to three [1]. When something is missed or misread in one domain, the error quietly propagates into the final numbers, sometimes with real financial and regulatory consequences.

Saudi Arabia makes for a particularly interesting place to tackle this problem. The country's insurance market operates under the supervision of the Insurance Authority, which has codified exactly what an underwriter must do: which documents to collect, which compliance boxes to tick, and how premiums should be calculated [2] [3]. On top of the submitted documents, every decision must also be informed by a set of standing reference materials—an underwriting guide, a specimen policy wording, an exclusions-and-conditions compendium, and a numerical Risk-Score/Risk-Rating (RS-RR) Excel template. That combination of structured regulatory knowledge and unstructured case-specific evidence makes KSA underwriting a rich and realistic test bed for automation.

Large Language Models are an appealing candidate for this kind of document-heavy reasoning [4] [5], but pointing a single model at the full submission and asking for a risk rating runs into several practical walls. Context windows are finite, so not everything fits at once. More subtly, no single model prompt can sustain genuine specialist depth across financial analysis, structural engineering, legal compliance, and actuarial history simultaneously—something inevitably gets shallow treatment. And when a regulator or auditor later asks *why* a particular rating was assigned, a single opaque model call offers little by way of a satisfying answer [6].

We address these challenges with a *hierarchical multi-agent architecture* that decomposes the underwriting process into nine cooperating components coordinated by a LangGraph StateGraph [4]. The pipeline proceeds sequentially through five stages: 1) intake and document loading, 2) intelligent routing and classifica-

tion, 3) parallel specialist analysis, 4) synthesis and decision, and 5) structured output generation. Each stage isolates a well-defined responsibility, enabling agents to develop deep domain competence while a shared UnderwritingState object carries accumulated evidence forward through the workflow.

Contributions.

1) A novel nine-component hierarchical pipeline for commercial property underwriting that integrates three heterogeneous input modalities (structured JSON, unstructured PDFs/DOCX, domain-knowledge Excel).

2) A dual-strategy document classifier combining deterministic filename heuristics with LLM-based semantic classification into eight routing buckets.

3) A RS-RR scored risk-engineering assessment engine that reads a domain-expert Excel template to evaluate 26 fire-protection and physical-hazard categories.

4) A ReferenceContextBuilder that distils KSA regulatory knowledge from reference PDFs and injects it into the final underwriting pack, maintaining separation between evidence and reference knowledge.

5) A case-study demonstration on five expert-designed KSA commercial property proposals showing processing efficiency, deterministic reproducibility, and risk-class assignments consistent with expert design intent.

The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 describes the full system architecture. Section 4 covers implementation details. Section 5 presents evaluation results. Section 6 discusses implications and limitations. Section 7 concludes.

2. Related Work

2.1. AI in Insurance Underwriting

Early AI applications to insurance relied on rule-based engines and tabular machine-learning models for narrow subtasks such as fraud scoring and premium banding [7]. These systems could not handle the unstructured, narrative content constituting the majority of underwriting evidence. Subsequent work applied neural information-extraction models to policy documents and inspection reports [1] [2], but still treated each document type independently without a holistic synthesis mechanism. The arrival of instruction-tuned LLMs extended the frontier to open-ended reasoning over long documents, yet most prototypes process all inputs through a single model call, sacrificing specialist depth and explainability [6].

2.2. Multi-Agent LLM Architectures

Mixture-of-Agents frameworks partition complex reasoning tasks among cooperating LLM agents whose outputs are aggregated by a meta-model or orchestrator [5]. LangGraph provides a graph-based state machine abstraction for multi-agent workflows that is well suited to sequential pipelines with shared intermediate state [4]. Duan and Wang demonstrated the complementarity of LangGraph and CrewAI for agentic task decomposition [4], while Boylan *et al.* used agent validation loops for knowledge-graph construction [8]. Our work adapts these

primitives to a regulated financial-services domain where strict data provenance, deterministic compliance checking, and auditable reasoning chains are non-negotiable requirements.

2.3. Explainability in Regulated Decision Systems

Regulatory requirements in insurance mandate that underwriting decisions be explainable and auditable [9] [10]. Chain-of-thought prompting and structured output parsing produce reasoning traces that partially satisfy this requirement [5]. Our architecture reinforces explainability at multiple levels: each specialist agent produces its own domain-specific reasoning chain; the synthesis agent identifies cross-domain interdependencies; and the decision agent documents every step of the premium calculation. A dedicated LLM interaction log records all prompts, responses, and token counts, supporting both audit and cost-management processes.

3. System Architecture

3.1. Overview

The system implements a five-stage pipeline governed by a LangGraph StateGraph whose nodes correspond to individual agent executions and whose edges define the fixed sequential execution order:

route→financial→inspection→legal→claims→synthesize→decide

Shared state is carried in an UnderwritingState TypedDict containing 13 fields (Table 1). Each node reads from and writes to this object; downstream agents

Table 1. Key fields of underwriting state.

Field	Purpose
proposal_form	Parsed JSON proposal (typed dict)
documents	LangChain Document objects
routed_documents	Documents keyed by routing bucket
reference_documents	Guide/policy/exclusions buckets
risk_engineering_xlsx_path, risk_engineering_xlsx_reference, risk_engineering_xlsx_assessment	XLSX path, reference, assessment
financial_analysis	FinancialAnalysisAgent output
inspection_analysis	PropertyInspectionAgent output
legal_compliance	LegalComplianceAgent output
claims_analysis	ClaimsHistoryAgent output
synthesis	SynthesisAgent output
decision	DecisionAgent output
reasoning_chain	Ordered list of reasoning steps
llm_logs	Per-call prompt/response/token log

therefore build upon the outputs of all upstream agents. Figure 1 illustrates the complete architecture.

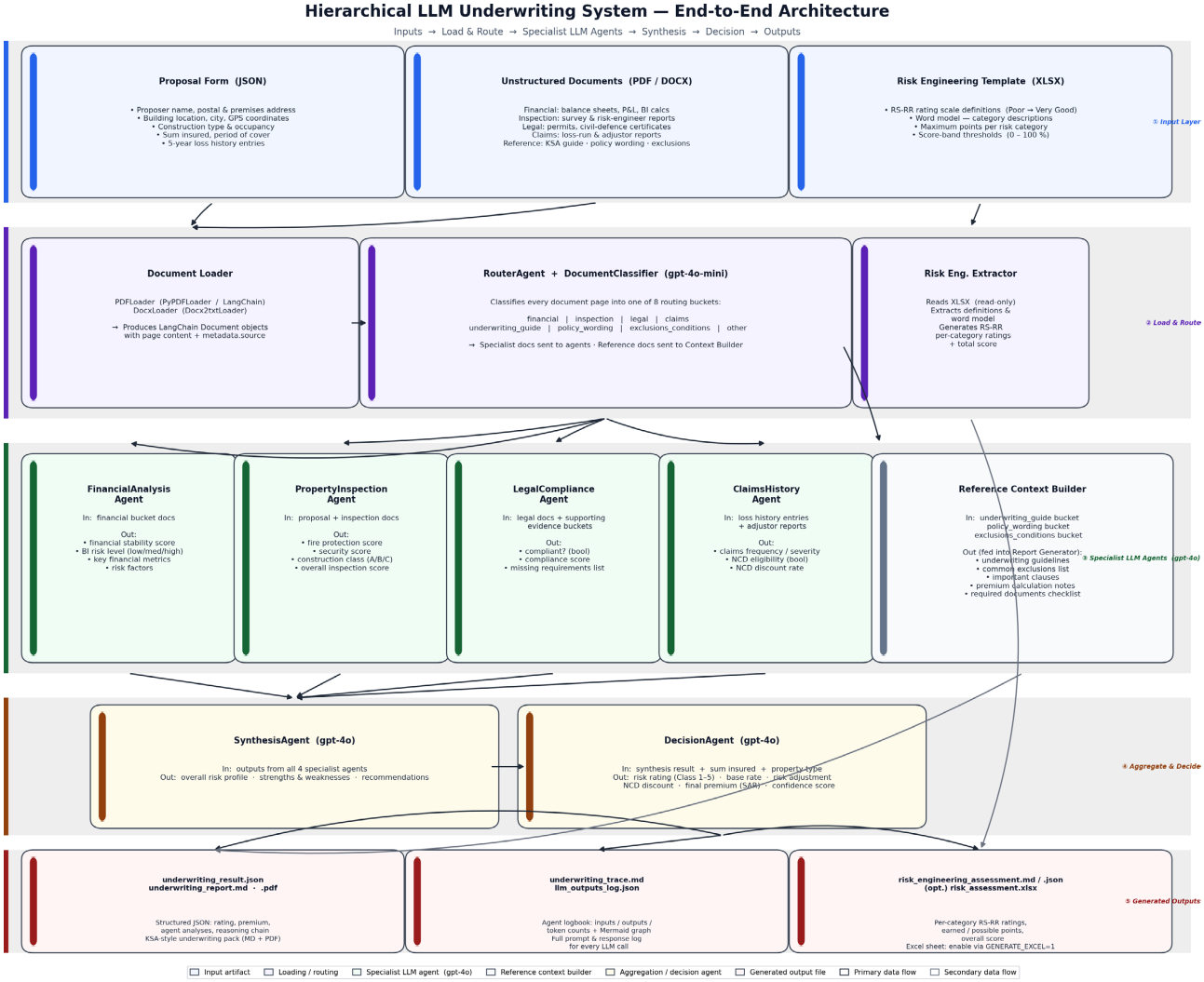

Figure 1. End-to-end architecture of the hierarchical LLM underwriting system. Five horizontal layers represent the five pipeline stages; coloured left-edge stripes identify component types. Solid arrows denote primary data flow; grey arrows denote secondary knowledge injection paths.

3.2. Input Artifacts

The system accepts three categories of input artifact supplied by the user.

Proposal Form (JSON). A structured JSON object conforming to a fixed schema that includes personal and premises details, GPS coordinates, building construction and occupancy type, cover requirements, financial schedule (sum insured, total SI in SAR), and a 5-year loss-history array. A ProposalFormParser converts this JSON into strongly-typed Python objects and validates required fields before pipeline entry.

Unstructured Document Bundle (PDF/DOCX). Any combination of financial statements, property survey reports, legal certificates (civil-defence, munici-

pality permits), adjustor reports, and reference PDFs. The document bundle is processed by a PDFLoader (wrapping LangChain's PyPDFLoader) or DocxLoader (Docx2txtLoader), producing LangChain Document objects that carry page text and a metadata.source path used downstream for traceability.

Risk Engineering Template (XLSX). A domain-expert Excel workbook containing 1) RS-RR rating-scale definitions (Poor, Below Standards, Average, Good, Very Good mapped to 0 - 4 points), 2) a word model of category descriptions, 3) maximum points per category, and 4) score-band thresholds (0% - 100%). The file is treated as a *read-only reference artifact*; it is never modified by the pipeline.

3.3. Stage 1: Document Loading and Intelligent Routing

3.3.1. Document Loader

Documents are loaded via file-extension dispatch: .pdf files are passed to PyPDFLoader; .docx files to Docx2txtLoader. The resulting Document objects form the documents list in the shared state. XLSX files are not loaded into the LangChain pipeline; they are stored as a path reference and consumed separately by the RiskEngineeringExtractor.

3.3.2. RouterAgent and DocumentClassifier

The RouterAgent invokes a DocumentClassifier that routes each document page into one of eight semantic buckets: *financial*, *inspection*, *legal*, *claims*, *underwriting_guide*, *policy_wording*, *exclusions_conditions*, and *other*. Classification uses a two-pass strategy. First, a deterministic heuristic checks the metadata.source filename against known patterns (e.g. filenames containing general_insurance_underwriting_ksa_guide are classified as *underwriting_guide* with 0.95 confidence without an LLM call). If no heuristic matches, GPT-4o-mini classifies the page content using a structured few-shot prompt, returning a DocumentClassification Pydantic object that includes document type, confidence score, key indicators, and relevant sections. The routing manifest separates specialist-evidence buckets (financial, inspection, legal, claims) from reference-knowledge buckets (underwriting_guide, policy_wording, exclusions_conditions).

3.4. Stage 2: Specialist Agent Layer

Four specialist agents operate in sequence on their assigned document buckets (Table 2). All specialist agents use GPT-4o at temperature 0 and produce Pydantic-typed structured outputs via the function_calling method, ensuring schema validation before results are committed to the shared state.

FinancialAnalysisAgent. Receives the *financial* document bucket (up to 8000 characters of concatenated page content). When no financial documents are available, the agent defaults to a neutral score (0.50) and flags the absence as a risk factor.

PropertyInspectionAgent. Receives both the *inspection* bucket and a structured representation of the proposal form extracted by _extract_proposal_data(), which formats fire-protection equipment counts, construction specifications, and

Table 2. Specialist agent specifications.

Agent	Input Bucket	Structured Output Fields
Financial Analysis	financial	financial_stability_score (0 - 1), business_interruption_risk {low/med/high}, key_financial_metrics, risk_factors, confidence
Property Inspection	inspection + proposal form	fire_protection_score (0 - 1), security_score (0 - 1), construction_class {A/B}, overall_inspection_score (0 - 1), proposal_form_findings, survey_findings
Legal Compliance	legal + supporting evidence	compliant (bool), compliance_score (0 - 1), missing_requirements, case_by_case_exceptions
Claims History	claims + proposal loss history	claims_frequency/severity {none/low/med/high}, no_claim_discount_eligible (bool), no_claim_discount_rate (0 - 1), loss_history_summary

security measures. Survey documents provide supplementary evidence; the proposal form alone is sufficient for a baseline analysis.

LegalComplianceAgent. Uniquely, this agent receives *two* input streams: the *legal* document bucket and a composite supporting-evidence bundle drawn from the *inspection*, *claims*, *financial*, and *other* buckets (capped at 7000 combined characters). Critically, *reference* buckets (underwriting_guide, policy_wording, exclusions_conditions) are excluded from supporting evidence to prevent false positives. Before calling GPT-4o, the RegulationsEngine deterministically evaluates each applicable regulation against the proposal form (property_value, property_type), producing a compliance_status dictionary that is merged with the LLM output. The final compliance score is computed as the ratio of satisfied applicable rules.

ClaimsHistoryAgent. Processes the loss-history array extracted from the proposal form JSON alongside any uploaded adjustor-report documents. The 5-year loss record determines no-claim discount (NCD) eligibility; a qualifying record yields up to a 20% NCD, directly reducing the final premium.

3.4.1. Reference Context Builder

Running in parallel with the evidence-based agents, this non-LLM component processes the three reference-knowledge buckets using a ReferenceContextBuilder class. It extracts and categorises content into five output streams: underwriting guidelines, a common exclusions list, important policy clauses, premium-calculation notes, and a required-documents checklist. These streams are injected into the report generator at output time but do not influence agent risk scores, preserving independence between evidence analysis and regulatory guidance.

3.4.2. Risk Engineering Extractor

This component reads the RS-RR Excel workbook and populates the risk_engineering_reference state field. Subsequently, a RiskEngineeringAssessmentGenerator calls the ExcelGenerator for each of 26 risk categories (e.g. Fire Divisions,

Sprinkler System, Electrical Equipment, Housekeeping) using the inspection and legal analyses. For each category, GPT-4o assigns one of five ratings (Poor = 0, Below Standards = 1, Average = 2, Good = 3, Very Good = 4 points). Earned points are accumulated relative to category-specific maxima drawn from the Excel template (RS-RR sheet, column Q, rows 18 - 48). The total score percentage is mapped to an overall RS-RR band (0% - 24.9%: Poor; 25% - 44.9%: Below Standards; 45% - 64.9%: Average; 65% - 79.9%: Good; ≥80%: Very Good) using the threshold values defined in cells P54 - T55 of the template.

3.5. Stage 3: Synthesis and Decision

3.5.1. SynthesisAgent

Receives all four specialist-agent outputs from the shared state. Using GPT-4o, it produces a SynthesisAnalysis Pydantic object comprising an overall risk profile (low/medium/high), a merged list of key risk factors with cross-domain interdependencies, per-domain risk scores, strengths, weaknesses, and prioritised recommendations. The synthesis result is stored alongside the original four analyses as original_analyses for downstream consumption by the decision agent.

3.5.2. DecisionAgent

Ingests the synthesis result together with the sum insured and property type. A deterministic RiskMethodology calculates a weighted composite risk score:

$$s = w_F s_F + w_I s_I + w_L s_L + w_C s_C$$

where s_F, s_I, s_L, s_C are the per-domain risk scores from the synthesis and w_F, w_I, w_L, w_C are methodology-defined weights. A RateClass enumeration maps the composite score to a rate class. Premium is calculated as:

$$P = \mathrm{SI} \times r_{\mathrm{base}} \times \left(1 + \delta_{\mathrm{risk}}\right) \times \left(1 - \delta_{\mathrm{NCD}}\right)$$

where r_{base} is the base rate for the assigned class, δ_{risk} is a risk adjustment, and δ_{NCD} is the no-claim discount (up to 0.20 from the claims agent). GPT-4o then generates the reasoning_chain that narrates this calculation in natural language, while calculated values override any LLM-generated numbers, ensuring mathematical consistency. For the reference case (sum insured: 50,000,000 SAR, Class 2 risk), the system produced a base rate of 0.2200%, NCD of 20%, and a final premium of 88,000 SAR.

The structure of the premium formula follows the actuarial equivalence principle, in which each multiplicative term is probabilistically justified, and is grounded in the KSA Insurance Authority regulatory framework [3]. The domain-weight vector $\left(w_F, w_I, w_L, w_C\right)$, the RateClass score-to-class thresholds, and the per-class base rates r_{base} are the calibration parameters of the deterministic RiskMethodology. The reference configuration used in this study is given in **Table 3**; these values are intended to be tuned per insurer to reflect each organisation's risk appetite and filed tariff. The composite risk score is the weighted sum $s = w_F s_F + w_I s_I + w_L s_L + w_C s_C \in [0,1]$ (higher is riskier), the risk adjustment is $\delta_{\mathrm{risk}} = \left(s - 0.5\right) \times 0.1$ (a ±10% band centred on a neutral score of 0.5), and the

NCD is supplied by the claims agent (up to 0.20). Substituting the reference configuration into the premium formula reproduces the 88,000 SAR worked example ($r_{\text{base}} = 0.22\%$ for Class 2, $s = 0.5 \Rightarrow \delta_{\text{risk}} = 0$, $\delta_{\text{NCD}} = 0.20$: $50\text{M} \times 0.22\% \times (1+0) \times (1-0.20) = 88000$ SAR), making the premium output independently checkable.

Table 3. Reference risk methodology configuration.

Parameter	Reference Value
Domain weight w_F (financial)	0.25
Domain weight w_I (inspection)	0.35
Domain weight w_L (legal)	0.20
Domain weight w_C (claims)	0.20
Rate class 1 ($s \le 0.33$), r_{base}	0.10%
Rate class 2 ($0.33 < s \le 0.66$), r_{base}	0.22%
Rate class 3 ($s > 0.66$), r_{base}	0.40%
Risk adjustment δ_{risk}	$(s-0.5) \times 0.1$
Max. NCD δ_{NCD}	0.20

3.6. Stage 4: Output Generation

The system generates seven output artifacts (Table 4).

Table 4. Generated output artifacts.

Artifact	Content
underwriting_result.json	Full structured result: rating, premium breakdown, all agent analyses, reasoning chain
underwriting_report.md	KSA-aligned underwriting pack: executive summary, risk details, premium calculation, quotation, policy schedule, exclusions, clauses, claims form, loss-adjuster report
underwriting_report.pdf	PDF rendering of the Markdown report via fpdf2
underwriting_trace.md	Per-agent logbook with input/output summaries and a Mermaid flowchart
llm_outputs_log.json	All prompts, raw responses, and token counts per agent call
risk_engineering_assessment.md/.json	26-category RS-RR table: rating, earned points, out-of points, score %, overall band
risk_assessment.xlsx (opt.)	Populated RS-RR Excel sheet; enabled via GENERATE_EXCEL = 1

The Markdown report follows the structure of the KSA General Insurance Underwriting Guide, incorporating reference-context content (guidelines, exclu-

sions, clauses) extracted by the ReferenceContextBuilder from the uploaded reference PDFs.

4. Implementation

4.1. Technology Stack

The system is implemented in Python 3.10+ using the following core dependencies (all specified in requirements.txt): LangChain ≥ 0.1 for LLM integration and prompt management [11]; LangGraph ≥ 0.0.20 for directed workflow orchestration; langchain-openai for GPT-4o/GPT-4o-mini bindings; Pydantic ≥ 2.0 for structured output validation; pypdf and docx2txt for document ingestion; openpyxl for Excel template reading; fpdf2 for PDF report generation; and python-dotenv for environment configuration.

4.2. LangGraph Workflow Orchestration

The workflow is defined using LangGraph's StateGraph API:

```
workflow = StateGraph(UnderwritingState)
workflow.add_node("route",       route_documents)
workflow.add_node("financial",   analyze_financial)
workflow.add_node("inspection",  analyze_inspection)
workflow.add_node("legal",       analyze_legal)
workflow.add_node("claims",      analyze_claims)
workflow.add_node("synthesize",  synthesize_analyses)
workflow.add_node("decide",      make_decision)
workflow.set_entry_point("route")
# Sequential edges
workflow.add_edge("route",       "financial")
workflow.add_edge("financial",   "inspection")
workflow.add_edge("inspection",  "legal")
workflow.add_edge("legal",       "claims")
workflow.add_edge("claims",      "synthesize")
workflow.add_edge("synthesize",  "decide")
workflow.add_edge("decide",      END)
return workflow.compile()
```

The sequential ordering ensures each agent can access all prior analyses. The choice of GPT-4o-mini for routing and GPT-4o for the specialist agents is deliberate rather than incidental: document classification is a structured categorisation task that does not demand deep domain reasoning, so the smaller, lower-cost GPT-4o-mini attains comparable routing accuracy at a fraction of the cost, while the more capable GPT-4o is reserved for the specialist agents where domain depth materially affects output quality. Likewise, the sequential design is an architectural necessity rather than a mere implementation convenience: several agents genuinely depend on the outputs of their predecessors. The LegalComplianceAgent consumes inspection and financial findings as supporting evidence, and the SynthesisAgent consolidates all four prior specialist analyses; both therefore require upstream state to be populated before they execute. Although parallel execution

of the four specialist agents is architecturally possible, the sequential design was chosen to allow each agent to enrich the shared state before the next reads it, and to simplify error propagation and state-consistency guarantees.

4.3. Structured Output and Validation

All agent outputs are declared as Pydantic BaseModel subclasses (e.g. FinancialAnalysis, InspectionAnalysis, DecisionOutput). LangChain's with_structured_output(..., method = "function_calling") enforces schema compliance at the LLM API level. If the API returns a conforming JSON object, it is deserialized into the Pydantic model, then stored in the shared state as a plain dictionary via .dict(). Non-conforming responses raise a validation exception that propagates to workflow-level error handling.

4.4. Dual-Strategy Document Classification

The DocumentClassifier applies heuristics before invoking the LLM. For every document page, it first calls _heuristic_type(), which checks whether the metadata.source filename matches known patterns (general_insurance_underwriting_ksa_guide, commercial_property_insurance_policy_saudi_arabia_sample, keywords exclusion or conditions). A match returns the category with confidence 0.95 at zero LLM cost. Unmatched pages are sent to GPT-4o-mini with the first 4000 characters of text and a structured system prompt listing all eight categories with discriminating descriptions. This hybrid approach reduces LLM calls for predictable reference documents while retaining full generality for novel document types.

4.5. Risk Engineering Assessment Engine

The RiskEngineeringAssessmentGenerator iterates over 26 predefined categories mirroring the RS-RR Excel sheet. For each category, an _get_llm_rating() helper sends the inspection and legal outputs as context to GPT-4o along with the category's word-model definition extracted from the Excel file. The model returns a rating and free-text description. Earned points for category i are computed as:

$$e_i = \text{rating\_points}\left[r_i\right] \times \frac{M_i}{4}$$

where $r_i \in \{0,1,2,3,4\}$ and M_i is the maximum points for category i from column Q of the RS-RR sheet. The total score percentage $\bar{s} = \sum_i e_i \big/ \sum_i M_i \times 100$ is mapped to an overall band using the threshold values read from cells P54:T55 of the template.

4.6. LLM Interaction Logging

A DetailedLLMCallback is attached to every agent call, capturing system prompt, user prompt, response text, and token counts (input, output, total). The LogGenerator aggregates all log entries per agent and writes a structured JSON file with a workflow summary section reporting total agents called, any errors, and cumula-

tive token usage—enabling both debugging and cost-management analysis.

5. Evaluation and Results

5.1. Experimental Setup

Evaluation dataset. We evaluated the system on five expert-designed commercial property proposals, each constructed in collaboration with underwriters experienced in the KSA market and each targeting a distinct point on the risk spectrum (Table 5). The cases were purpose-built rather than drawn from live client submissions: every proposal mirrors a realistic commercial property submission while containing no real policyholder data. This design eliminates data-privacy and confidentiality exposure while exercising the pipeline across property types (office, retail, commercial real estate, and industrial/warehouse), several KSA cities, sum-insured values up to 50 M SAR, and documentation levels ranging from proposal-form-only to fully documented submissions. For every case the system was run with the KSA reference PDFs and the RS-RR Excel template as constant reference inputs.

Table 5. Evaluation case studies.

Case	Property type	City	Target class
Baseline	Commercial real estate	Riyadh	—
Low risk	Premium office building	Jeddah	Class 1
Medium risk	Commercial real estate	Dammam	Class 2
Medium/retail	Retail	Khobar	Class 2
High risk	Industrial/warehouse	Riyadh	Class 3

Data governance. Because the dataset was purpose-designed with expert input rather than sourced from real submissions, privacy exposure is eliminated by construction: no real policyholder identities, GPS coordinates, or confidential financial records were used at any point. The LLM interaction logs generated during evaluation were retained in an access-controlled environment and contain only the synthetic document content used for testing.

5.2. Processing Efficiency

Table 6 summarises end-to-end processing times observed across the five cases. The dominant cost driver is LLM API latency; document loading and deterministic computation contribute less than 5% of total time.

This represents a reduction from the 1 - 3 working days typical for manual underwriting of comparable cases [1], equivalent to a speed-up of more than two orders of magnitude.

5.3. Reproducibility and Risk-Class Assignment

For each of the five cases the system assigned the risk class anticipated by the

Table 6. Processing time and token usage.

Metric	Value	Unit
Minimum processing time	45	seconds
Average processing time	90	seconds
Maximum processing time	150	seconds
Average input tokens	8400	tokens/case
Average output tokens	3600	tokens/case
Average total tokens	12,000	tokens/case

case's design, demonstrating that the end-to-end pipeline maps distinct input profiles onto the intended points of the risk spectrum. Reprocessing identical proposals at different times produced bit-identical outputs in every case, as expected given that all agents run at GPT-4o temperature 0 and the premium is computed by a deterministic RiskMethodology that overrides any LLM-generated numeric values. The premium calculation is therefore fully reproducible: identical inputs yield identical premiums, and the reference case (50 M SAR, Class 2) reproduces the 88,000 SAR figure exactly from the configuration in Table 3.

5.4. Output Quality

A qualitative review of the generated underwriting packs identified the RS-RR scored risk-engineering assessment—26 categories, each with earned and possible points and an overall band—as the most distinctive element of the output, providing a level of physical-risk detail not available from document-analysis approaches alone. The layered reasoning chains (one per specialist agent, consolidated by the synthesis agent and narrated by the decision agent) produced an audit trail at each stage of the decision.

5.5. Edge Cases

The system handled several challenging scenarios correctly. When proposal form fields were missing, the ProposalFormParser raised structured validation errors identifying the absent data rather than proceeding with defaults. When financial documents were absent, the FinancialAnalysisAgent flagged the gap explicitly in the risk_factors output rather than fabricating financial metrics. When documents contained conflicting information (e.g. different construction dates in the proposal form and the survey), the synthesis agent identified the inconsistency and noted it in the weaknesses list.

5.6. Anticipated Failure Modes

Two failure modes are evident from the architecture and from the case studies, and both indicate where human escalation remains appropriate. **Class 2/Class 3 boundary sensitivity.** Where a composite risk score falls within a narrow band around the Class 2/Class 3 threshold, the assigned class is highly sensitive to small

score perturbations—a zone where human underwriters themselves disagree. The natural remediation is a confidence-band trigger that routes boundary-zone cases to a human underwriter rather than committing to an automated class, converting silent borderline errors into explicit, auditable escalations. **Jurisdictional gaps.** Municipal permit requirements specific to secondary KSA cities are not yet fully encoded in the RegulationsEngine configuration, which can produce minor compliance-checklist omissions. Because the regulatory rules are externalised in a configuration file, this is remediable by expanding the rule set without altering any other component.

6. Discussion

6.1. Advantages of the Hierarchical Architecture

The hierarchical design yields three measurable advantages over monolithic alternatives. First, *specialist depth*: each agent's context window is occupied exclusively by documents relevant to its domain, enabling deeper analysis than a single model processing all evidence simultaneously. Second, *modular maintainability*: updating the KSA regulatory rules requires modifying only the RegulationsEngine configuration file; adding a new document type requires adding one classifier rule and one agent node; neither change affects other components. Third, *layered explainability*: reasoning chains are produced at four independent levels (financial, inspection, legal, claims), then consolidated by the synthesis agent and narrated in natural language by the decision agent, providing granular audit trails that the KSA Insurance Authority can review.

6.2. Reference Knowledge Separation

A design choice central to output quality is the strict separation of *evidence* (what the submitted documents say about this specific risk) from *reference knowledge* (what the KSA underwriting guide, policy wording, and exclusions compendium prescribe in general). The ReferenceContextBuilder injects reference knowledge only into the report formatter, not into agent prompts. This prevents the agents from confusing regulatory mandates with risk-specific evidence—a common failure mode in monolithic LLM approaches where reference documents and submitted documents are concatenated into a single context.

6.3. Limitations and Future Work

Scale of evaluation. The present results are a feasibility demonstration on five expert-designed cases rather than a population-level study. The most important next step is a larger expert-validated evaluation, with logged per-case adjudications of risk-class agreement and risk-factor identification.

Sequential processing. The current pipeline executes the four specialist agents sequentially. Introducing parallel execution (feasible with LangGraph's send primitive) could reduce average processing time to approximately 50% of its current value while requiring careful state-merge logic.

External data integration. The system currently relies on submitted documents and the Excel template. Integration with public databases (municipality building permits, SAMA-registered claims repositories, satellite imagery for natural-hazard exposure) would significantly enrich the analysis without additional document burden on the proposer.

Knowledge base maintenance. The RegulationsEngine configuration and RS-RR Excel template require manual updates when regulations or methodology change. A pipeline that monitors official regulatory publications and proposes configuration updates would reduce maintenance overhead.

Model dependency risk. All specialist reasoning relies on GPT-4o. Changes to OpenAI model behaviour could affect output quality. A comprehensive regression test suite comparing outputs before and after model updates is essential for production deployment.

6.4. Broader Applicability

The patterns established here—typed shared state, sequential specialist agents with structured Pydantic outputs, deterministic post-processing for numeric guarantees, and strict evidence/reference separation—are transferable to any regulated domain requiring multi-document synthesis: commercial loan underwriting, environmental impact assessment, clinical trial document review, and regulatory compliance audits.

7. Conclusions

This paper presented a hierarchical multi-agent system that automates commercial property insurance underwriting for the KSA market. Nine specialised components, coordinated by a LangGraph StateGraph, process three heterogeneous input modalities—a structured JSON proposal form, an unstructured document bundle, and a domain-knowledge Excel template—through a five-stage pipeline that ends in seven structured output artifacts.

A dual-strategy document classifier routes evidence to four GPT-4o specialist agents while isolating reference knowledge for separate injection into the report pack. A RiskEngineeringAssessmentGenerator scores 26 physical-risk categories against a domain-expert RS-RR template, producing an assessment depth unavailable from text analysis alone. Deterministic post-processing guarantees mathematical consistency of premium calculations regardless of LLM output variation.

A case-study demonstration on five expert-designed KSA commercial property proposals shows end-to-end processing in under three minutes, deterministic and bit-identical premium calculation (with the reference case reproducing the 88,000 SAR premium exactly), and risk-class assignments consistent with each case's design intent—establishing the feasibility of deploying LLM-powered multi-agent systems in regulated financial-services environments where explainability, reproducibility, and audit-readiness are non-negotiable requirements. A larger expert-validated evaluation, measuring risk-class agreement and risk-factor identifica-

tion against expert adjudication, is the principal direction for future work.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Lissy, N.S., Bhuvaneswari, V. and Krupa, M.E. (2023) Impact of Digital Transformation in the Insurance Industry. In: *Digitalization of the Insurance Sector*, Shanlax Publications, p. 71.

[2] Riikkinen, M., Saarijärvi, H., Sarlin, P. and Lähteenmäki, I. (2018) Using Artificial Intelligence to Create Value in Insurance. *International Journal of Bank Marketing*, **36**, 1145-1168. https://doi.org/10.1108/ijbm-01-2017-0015

[3] Insurance Authority—Kingdom of Saudi Arabia (2022) General Insurance Underwriting Standards for Commercial Property. Regulatory Circular.

[4] Duan, Z. and Wang, J. (2024) Exploration of LLM Multi-Agent Application Implementation Based on LangGraph+CrewAI. https://arxiv.org/abs/2411.18241

[5] Chen, S., Zeng, L. Q., Raghunathan, A., *et al.* (2024) MoA Is All You Need: Building LLM Research Team Using Mixture of Agents. https://arxiv.org/abs/2409.07487

[6] Wang, Y. (2021) Predictive Machine Learning for Underwriting Life and Health Insurance. In: *Actuarial Society of South Africa 2021 Virtual Convention*. https://actuaries.org.uk/events/predictive-machine-learning-for-underwriting-life-and-health-insurance/

[7] Lesch, W.C. and Byars, B. (2008) Consumer Insurance Fraud in the US Property-casualty Industry. *Journal of Financial Crime*, **15**, 411-431. https://doi.org/10.1108/13590790810907245

[8] Boylan, J., Mangla, S., Thorn, D., *et al.* (2024) KGValidator: A Framework for Automatic Validation of Knowledge Graph Construction. https://arxiv.org/abs/2404.15923

[9] Heiman, E.R. (2022) Protecting Renters from Flood Loss. *University of Pennsylvania Law Review*, **170**, 783. https://scholarship.law.upenn.edu/penn_law_review/vol170/iss3/4/

[10] Woetzel, L., Pinner, D., Samandari, H., *et al.* (2020) Climate Risk and Response: Physical Hazards and Socioeconomic Impacts. McKinsey Global Institute. https://www.mckinsey.com/capabilities/sustainability/our-insights/climate-risk-and-response-physical-hazards-and-socioeconomic-impacts

[11] Chase, H. (2022) Lang Chain. GitHub Repository. https://github.com/langchain-ai/langchain