Dingl AI

Software Requirements Specification (SRS)

Purpose

Dingl AI is an AI-powered enterprise platform for intelligent data ingestion and workflow orchestration. This SRS defines the requirements for Dingl AI's software system, including its functional capabilities and constraints. The purpose of Dingl AI is to **automate the ingestion and analysis of multimodal documents** (text, images, videos) and to orchestrate business workflows with minimal human intervention. By specifying these requirements, we ensure the system meets enterprise needs for document processing, process intelligence, and traceability in high-compliance B2B environments.

Product Scope

Dingl AI is designed to ingest large volumes of unstructured data (e.g. PDFs, scanned images, video transcripts) and transform them into structured, actionable insights. The product scope includes: **document and image OCR extraction**, intelligent data linking via a graph database, and answering complex user queries through integrated LLM agents. The system will support **user-defined pipeline configurations** for processing data, real-time monitoring dashboards, and integration points (APIs) to embed insights into other applications. Dingl AI targets B2B firms requiring **high traceability of documents and field data**, such as supply chain certification management, compliance auditing, and operations where document relationships must be preserved. The platform will be delivered as both a cloud SaaS and on-premises solution to accommodate different enterprise deployment needs.

Functional Requirements

1. **Multimodal Data Ingestion:** Dingl AI shall ingest documents (PDF, Word), images (scans, photos), and video/audio files. For image and video inputs, the system shall apply OCR and transcription to extract text content automatically.
2. **Pipeline Configuration:** The system shall allow users to **configure processing pipelines** through a UI or configuration files. Users can define a sequence of operations (e.g. OCR, text chunking, embedding, knowledge graph linking, validation rules) to customize how their data is processed.
3. **Content Chunking and Embedding:** Dingl AI shall break down large documents or video transcripts into manageable chunks and generate vector embeddings for each chunk. These embeddings will be stored in a vector database (Qdrant) for semantic similarity searchpython.langchain.com.
4. **Knowledge Graph Extraction:** The system shall extract key entities and relationships from documents (e.g. buyer-seller relationships, document metadata) and store them in a graph database (Neo4j). This provides a knowledge graph that retains relationships between data points for advanced querying (e.g. linking a certificate to related shipments).
5. **Rule-Based Validation:** The system shall support rule-based checks on ingested data. For example, it should flag discrepancies (such as shipment date earlier than issue date, or mismatched totals) using predefined rules, ensuring data consistency and compliance.
6. **Retrieval-Augmented Q&A:** Dingl AI shall enable users to ask natural language questions against the ingested data. An LLM agent will retrieve relevant information from the vector store and knowledge graph, then generate answersqdrant.techqdrant.tech. This allows interactive document querying and process intelligence (e.g. *"Show all certificates pending approval and their issuers."*).
7. **Real-Time Monitoring & Alerts:** The system shall provide a dashboard that displays pipeline processing status in real-time. It will show metrics such as number of documents processed, processing duration, and any errors. Users shall receive alerts (e-mail or in-app) if a pipeline fails or if a validation rule triggers, enabling timely interventions.
8. **Reporting and Analytics:** Dingl AI shall produce summary reports (periodic or on-demand) of processed data – for example, monthly counts of documents ingested, trends in data fields, or KPI scorecards. These reports should be accessible through the UI and exportable (CSV/PDF) for audit purposes.
9. **Fine-Tuning Support:** The platform shall allow integration of custom AI models and fine-tuning. Users (with appropriate privileges) can provide additional training data or feedback, and the system will fine-tune the underlying LLM (via

the Ollama engine) to improve domain-specific accuracy. This ensures the AI models can be adapted to the organization's terminology and document patterns.

10. **External API Integration:** Dingl AI shall expose a RESTful API enabling external applications to interact with it. This includes endpoints to submit documents for ingestion, query the knowledge base (e.g. search or Q&A), and retrieve results or reports. The API will use secure authentication (e.g. API keys or OAuth) so that enterprise IT systems can integrate Dingl AI's capabilities into their workflows.
11. **User Management and Access Control:** The system shall support multiple user roles (e.g. Administrator, Data Engineer, Analyst). Permissions must be configurable so that, for instance, only admins can change pipeline configurations or fine-tune models, while analysts can view results and run queries. All user actions should be logged for auditability.
12. **System Administration:** Dingl AI shall provide admin functions such as scheduling pipeline runs (e.g. nightly batch ingestion), adjusting configurations for connectors (e.g. linking to data sources like cloud storage or databases), and health monitoring of system components. It should also allow backup and restoration of the vector database and graph database data.

Non-functional Requirements

- **Performance:** The system should handle high-volume data processing. It is expected to process on the order of thousands of pages of documents per hour. Query response time for the knowledge base (vector + graph lookup and LLM answer generation) should ideally be within a few seconds for typical queries. Batch ingestion pipelines should scale by parallelizing tasks (e.g. OCR and embedding can be distributed across available compute).
- **Scalability:** Dingl AI must scale both **vertically** (leveraging more powerful machines with GPUs for faster OCR/LLM processing) and **horizontally** (multiple pipeline workers, clustering of the vector DB, etc.) to accommodate growing data loads. The design shall be cloud-native, allowing deployment to Kubernetes or similar for auto-scaling in the SaaS model.
- **Security:** Given enterprise usage, robust security is required. Data at rest in the vector store and graph store shall be encrypted. In transit, all APIs and data flows use HTTPS/SSL. Role-based access control and auditing are mandatory for all user interactions. On-premise deployments will run in the client's secure network environment, and the system must not transmit sensitive data externally without permission.
- **Privacy Compliance:** The system should facilitate compliance with data protection regulations. For example, it should support data retention policies (to

purge or anonymize data after a certain period) and allow extraction of all data related to an entity for compliance audits. No personal data is used for model training without consent, and any use of external LLM APIs will be configurable to disable if privacy is a concern.

- **Reliability & Availability:** Dingl AI should be designed for high availability, especially in production (target 99.5% uptime for the cloud service). It should gracefully handle failures: e.g. if the vector database or LLM service is temporarily down, the system queues tasks and recovers without data loss once services resume. Transaction integrity is important — for each document, either all its data (vector embedding, graph entries) are saved or none in case of error, to avoid partial ingestions.
- **Maintainability & Modularity:** The software architecture shall be modular, separating concerns (e.g. ingestion pipeline, data stores, UI, LLM service) to ease maintenance and upgrades. Components like the OCR module or LLM engine should be swappable with minimal changes (for instance, using a different OCR library or upgrading to a newer LLM). Clear logging and error messages are required to aid in troubleshooting issues. The codebase will follow standard coding conventions and include documentation for pipeline configurations, making it easier for developers to extend or modify behavior.
- **Usability:** The user interface (for the dashboard and configuration) should be intuitive and require minimal training. Pipeline configuration might be presented in a low-code manner (like a visual flow builder or YAML templates with examples). Users should be able to quickly see the status of their data (e.g. how many documents ingested, any errors) and navigate to details. The system will support English at launch, with a plan to support localization for other languages if needed by clients.
- **Compatibility:** Dingl AI's components are built with modern open-source tech (Python, containerized databases, etc.), ensuring it can be deployed on standard enterprise Linux servers or cloud VMs. The system should work in both cloud and air-gapped on-premise environments. For on-premise, any external dependencies (like model downloads) should have an offline installation option.
- **Extensibility:** The design should allow adding new data processing modules (e.g. a new file type parser or a new machine learning model) without major refactoring. For instance, if a client later needs to process sensor IoT data in the workflow, a module for that input can be integrated with the existing pipeline manager.

External Interfaces

User Interface: Dingl AI will provide a **web-based dashboard** for users. Through this interface, users can upload or link documents for ingestion, configure pipelines (via forms or by uploading config files), and view processing results and analytics. The UI will include visualization of the knowledge graph (e.g. an interactive graph view to explore linked entities) and tools to query the data (such as a search bar or chat interface for the LLM Q&A).

Application Programming Interface (API): All major functionalities will be accessible via RESTful APIs. For example:

- POST /ingest – to submit documents or data for ingestion.
- GET /query – to query the knowledge base with a question or search term.
- GET /results/{id} – to retrieve processed results or reports.
- POST /fine-tune – to submit data for model fine-tuning (privileged endpoint). This allows integration with enterprise systems (ERP, CRM, etc.) so Dingl AI can receive data automatically or provide answers within other applications. The API returns data in JSON format and uses standard HTTP response codes.

Integration Interfaces: Dingl AI will interface with several external systems and components:

- **Vector Database (Qdrant):** The system interacts with Qdrant via its Python client or HTTP API for vector insertion and similarity searchpython.langchain.com. This is largely an internal interface, but configuration (like Qdrant host URL, index names) will be exposed for admin settings.
- **Graph Database (Neo4j):** Communication with Neo4j is done through its Bolt protocol or REST API. The system will use Neo4j drivers to upsert nodes/relationships and to query graph patterns. External tools could also query Neo4j directly if granted access, for advanced graph analytics.
- **LLM Engine (Ollama):** Dingl AI uses the Ollama service to host large language models locally. The pipeline manager communicates with the Ollama LLM server (e.g. via HTTP endpoints provided by Ollama's runtime) to generate text (answers or to perform NLP tasks)medium.com. In a cloud setup, the LLM might be served via a microservice. This interface is mainly internal, but the model selection (which model to use, fine-tune, etc.) can be configured by an admin.
- **Authentication/SSO:** For enterprise integration, the system will support SSO via SAML or OAuth2.0 so that corporate users can log in with their existing

credentials. This interface will connect to corporate identity providers (e.g. Azure AD, Okta).

- **Data Sources:** Optionally, Dingl AI can interface with external data sources (if configured) such as cloud storage buckets, email servers, or databases to automatically ingest files. For instance, a connector might periodically fetch new files from an SFTP server. These connectors will use appropriate protocols (SFTP, AWS SDKs, etc.) and are configured externally.

All external interfaces are documented with their usage and expected inputs/outputs. The system will include **API documentation** for developers and a user manual for the UI.

Use Case Narratives

Use Case 1: Automated Certificate Processing
Goal: Ingest and validate supply chain certificates for compliance.
Actor: Compliance Officer (user); Dingl AI system.
Scenario: A compliance officer uploads a batch of trade certificates (PDFs) into Dingl AI. The system's pipeline automatically extracts text from each PDF (using OCR for scanned ones) and parses key fields (buyer, seller, issue date, etc.). It then runs rule checks (e.g. ensuring *ShipmentDate* is not earlier than *IssueDate*). The vector database is updated with the textual content, and the relationships (e.g. *Company A* issued *Certificate X* for *Company B*) are stored in the graph DB. The officer opens the dashboard to monitor progress and sees all certificates processed. One certificate is flagged due to a rule violation – the dashboard highlights it, and the officer clicks to see details of the error (R1: Shipment date precedes issue date). The officer can then take corrective action in the source data. Later, the officer queries the system: *"List all certificates issued by Company A in 2023 and their shipment statuses."* The LLM agent retrieves relevant certificates via semantic search and graph lookup, providing an answer within seconds. The officer obtains an aggregated report instead of manually searching through dozens of documents – a task that would have taken hours is now done in minutes.

Use Case 2: Field Inspection Workflow Orchestration
Goal: Integrate field image data and orchestrate a review workflow.
Actor: Operations Manager; Dingl AI system; Field Engineer (data source).
Scenario: A field engineer captures images and notes from equipment inspections using a mobile app. These images and notes are sent to Dingl AI via the API. Dingl AI's multimodal pipeline extracts text from the notes and uses computer vision (if configured) to detect certain objects or labels in images (e.g. reading serial numbers or gauge readings from photos). All extracted data is indexed. The operations manager has

configured a pipeline such that if any critical issues are mentioned (e.g. the word “fault” or a measurement out of range), Dingl AI automatically triggers an alert. When an image indicating a faulty component is ingested, the system identifies the issue and, via its integration, creates a case in the maintenance tracking system (through a webhook API call). The manager receives a Dingl AI alert and sees a new case has been opened with relevant data. The manager can then query Dingl AI: *“Show me all past inspection reports for this equipment and any related issues.”* The system uses the knowledge graph to pull up historical records and summarizes them with the LLM. This **agent-based orchestration** streamlines what used to require manual coordination across systems, improving response time and data accuracy in the workflow.

Use Case 3: Knowledge Discovery and Q&A
Goal: Retrieve insights from a large collection of documents (knowledge base) using natural language.
Actor: Business Analyst; Dingl AI system.
Scenario: A business analyst needs to gather insights from thousands of pages of technical documents (manuals, reports, contracts). They have all been ingested into Dingl AI over time. The analyst goes to the Dingl AI dashboard and asks a question in the chat interface: *“What are the key differences between Policy X and Policy Y in terms of data retention?”*The system breaks this query into parts, uses the **LangGraph agent orchestration** to decide it should search the vector store for Policy X and Y documentsqdrant.tech, and finds the relevant sections. The LLM then generates an answer, citing the differences found. The analyst refines the query: *“List the documents where Policy X was referenced after 2022.”* The system queries the graph DB for documents linked to Policy X, filters by date, and uses the LLM to format a concise list. The analyst quickly gets answers without reading the documents manually. This showcases Dingl AI’s ability to **combine semantic search and graph-based filtering** on user queries, providing a powerful research assistant.

Constraints

- **Technology Constraints:** The solution is built specifically with certain technologies: Qdrant for vector search, Neo4j for graph storage, and the Ollama framework for hosting LLMs. These choices imply constraints such as requiring Python-compatible environments and possibly GPU support for the LLM. For on-premise deployments, the client’s hardware must be capable (e.g. enough RAM/CPU, and optional GPU for faster model inference). If a customer’s policy disallows cloud services, they must use the on-prem version because Dingl AI’s

design currently doesn't support other cloud vector databases (no substitution for Qdrant without significant changes).

- **Regulatory and Policy Constraints:** Dingl AI must operate within data governance rules of clients. For example, in finance or healthcare sectors, data can't leave certain geographic regions. This constrains the deployment – the SaaS cloud offering might not be acceptable to some clients, necessitating on-premise installation. Also, the system's machine learning components must be explainable to a degree (for compliance), which constrains how black-box the LLM usage can be; we may need to log and justify outputs for audit.
- **Timeline Constraints:** The project follows a pilot-first deployment with a planned GA (General Availability) release in 6–10 months. This imposes a constraint on scope for the pilot: initially, not all features will be fully scaled. For example, fine-tuning might be available in pilot as a beta feature with limited model sizes, and more extensive support will come by GA. The system design must accommodate iterative development – e.g., an initial simple UI in the pilot, enhanced by GA.
- **Interoperability Constraints:** The platform must integrate with existing enterprise systems (ERP, CRM, etc.). It should use standard data formats and protocols (JSON, CSV, REST), but any deviation or proprietary format would be a constraint. For instance, if a client needs it to interface with an older system only supporting XML/SOAP, that's a constraint to handle separately or via an adapter.
- **User Constraints:** End-users might not be AI experts. The system's configuration (like writing prompt templates for the LLM, or setting threshold for similarity search) must have sensible defaults. We assume users won't fine-tune those without guidance. This constrains the design to provide pre-tuned parameters and possibly an **auto-configuration** mode to simplify user experience.
- **Maintenance & Support Constraints:** For on-prem deployments, updates cannot be continuous (as in SaaS) due to client change control policies. This means Dingl AI on-prem releases will be packaged and perhaps updated quarterly. The architecture must support backward compatibility and smooth upgrade paths under these constraints. Also, not all clients will allow sending telemetry or usage data back to developers, constraining how we gather feedback (we rely on in-depth user testing and manual logs review instead).

Overall, these constraints shape Dingl AI's implementation and delivery. Despite them, the system is designed to be robust and flexible within the given boundaries, ensuring it addresses the core needs of enterprise document and process intelligence.

Figure: High-level system overview of Dingl AI, showing user interaction through a Dashboard/API, the Pipeline Manager orchestrating data flow to the Vector DB (Qdrant), Graph DB (Neo4j), and LLM engine (Ollama) for analysis.

Figure: Data ingestion and processing flow. Documents/images/videos are processed through OCR and NLP, then stored as embeddings in Qdrant and as linked entities in Neo4j. Monitoring and dashboards provide real-time insights.

Dingl AI – Software Design Description (SDD)

System Overview

Dingl AI's architecture follows a **modular, agent-based design** to handle multimodal data processing and intelligent orchestration. At a high level, the system is composed of the following major subsystems: an **Ingestion Pipeline Manager**, a **Vector Index (semantic memory)**, a **Graph Database (knowledge graph)**, an **LLM Orchestration Engine**, and a **User Interface & API layer**. The system ingests raw data (documents, images, etc.), processes and enriches it, then stores results in the vector and graph stores. When users query the system or when automated workflows are triggered, an AI agent (LLM-driven) orchestrates retrieval from these stores and generates outputs.

Key design principles include **loose coupling** between components and **agent-based orchestration**. The pipeline manager (think of it as the conductor) does not hardcode a single linear process but instead can invoke different "agents" or modules depending on context. For example, an OCR agent handles images, a transcription agent handles video, a chunking agent splits text, etc., all coordinated by the manager. This aligns with an **Agentic RAG (Retrieval-Augmented Generation)** approach where multiple steps and data sources are combined to answer complex queriesqdrant.tech.

The architecture must support both **cloud-based (multi-tenant SaaS)** and **on-premises (single-tenant)** deployments:

- In the **SaaS model**, components like Qdrant and Neo4j run in a scalable cloud cluster, and the LLM (Ollama or alternative) runs in a containerized microservice. The pipeline manager is deployed as part of a cloud service that

can handle requests from multiple client organizations (with strict data isolation).

- In the **On-Prem model**, all components run on the client's servers. The design uses containerization (Docker/Kubernetes) so the whole system can be deployed behind the client's firewall. The architecture remains the same logically, but all calls (e.g., to the LLM or databases) are local.

In both scenarios, Dingl AI maintains a **modular design** where each service (ingestion, vector DB, graph DB, UI, LLM) can be independently updated or scaled. The system overview diagram below illustrates these components and their interactions.

Figure: Dingl AI System Overview – The user interacts via a dashboard or API, configuring pipelines and querying the system. The Pipeline Manager orchestrates processing, storing vectors in Qdrant and relationships in Neo4j, and leveraging the Ollama LLM for advanced analysis or responses.

Architectural Design

1. Ingestion Pipeline Manager

Responsibilities: Coordinates the end-to-end data processing workflow. It receives raw input (files or data references), then invokes specialized modules in sequence or in parallel according to a pipeline definition. Key sub-components and their functions:

- **Connector/Loader:** Handles acquiring input data. For uploaded files, it simply accepts them; for remote sources, it might include connectors (S3 downloader, email reader, etc.). This module yields raw content (e.g., PDF bytes or image frames).
- **OCR Module:** If input is an image or PDF with no text layer, this module uses an OCR engine (e.g., Tesseract or an ML-based OCR) to extract textstraitsresearch.com. For videos, a speech-to-text sub-module produces transcripts.
- **Parsing & Chunking Module:** Takes raw text (from OCR or directly from text-based docs) and splits it into logical chunks. This uses NLP (Natural Language Processing) techniques (sentence segmentation, maybe semantic segmentation). It may also parse structured data (tables, key-value pairs) if needed. Each chunk is typically a few sentences or a paragraph, suitable for embedding.

- **Embedding & Vector Indexing:** Invokes the embedding model (via the LLM engine or a dedicated embedding service) to convert each text chunk into a vector. These vectors, along with metadata (document ID, page number, etc.), are stored in the Qdrant vector database. This provides semantic search capabilities – similar texts can be found via nearest-neighbor search in the vector spacepython.langchain.com.
- **Knowledge Graph Populator:** Extracts key entities and relations. This could be done by regex rules or by using an NLP entity recognizer or even the LLM in analysis mode. For example, from a certificate document, it might extract "Buyer: X", "Seller: Y", "IssueDate: Z" and then create nodes (Buyer:X, Seller:Y, Certificate doc) and relationships (Certificate -> issued_by -> Seller). These are written to Neo4j. Neo4j's property graph model is used to store nodes (documents, companies, etc.) and relationships explicitlyneo4j.com. The design of this module is schema-driven: we define which entity types and relations we care about (perhaps configured via a schema file or UI), so it knows what to extract.
- **Rule Engine:** After populating data, the pipeline manager calls the rule engine to run any business rules on the new data (as described in SRS). This is a lightweight component where each rule is essentially a function or expression evaluated against the extracted data. Any rule violations are logged and sent to the monitoring/alerts system.
- **Output Compiler:** Finally, the pipeline manager compiles an output summary for that ingestion batch – listing processed items, any errors or flags, and storing a reference to all data (like vector IDs or graph entries) for quick retrieval. This is saved in an index or log (which the UI can display).

The Pipeline Manager is implemented such that each of these steps can be performed by an "agent" which could be a separate process or function call. In a cloud deployment, some heavy tasks (like OCR or embeddings) might be offloaded to worker containers (for scalability), communicating via an internal queue. In on-prem, they might just be threads or processes on the same machine.

Crucially, the pipeline manager is **configurable**: using a pipeline definition (in JSON/YAML or via UI), one can specify which modules to run, in what order, and with what parameters. For instance, some clients might skip knowledge graph building, so that step can be toggled off; others might add a custom step (e.g., calling an external API for data enrichment at a certain point – the design allows plugin modules). This flexibility is a cornerstone of the design, making Dingl AI applicable to various workflows.

2. Data Storage Components

Vector Database (Qdrant): Qdrant is a core component storing high-dimensional vectors for each data chunk. We chose Qdrant for its **extended filtering support and production-ready performance**python.langchain.com. The system interacts with Qdrant through its API to:

- Insert new vectors during ingestion (with payload metadata like doc_id, chunk_index, text).
- Search for similar vectors given a query vector (used during retrieval/Q&A).
- Filter queries by metadata if needed (e.g., search within a specific document or date range by using Qdrant's payload filters).

We designate separate collections in Qdrant for different data domains if needed (e.g., one collection per client organization in SaaS, or per document type). Qdrant runs as a separate service; in deployment, it might be containerized. The design ensures if Qdrant is down or unreachable, the pipeline manager will queue embeddings and retry – this decoupling is handled by a message queue or retry logic to not lose data.

Graph Database (Neo4j): Neo4j stores the knowledge graph. Its purpose is to maintain explicit **traceability and relationships** that might be hard to infer from raw text alone. For example, the fact that *Document A* corresponds to *Order 123* and is *approved by John Doe* can be explicitly represented. This enables queries like "find all documents approved by John Doe for Order 123" using graph traversal, which might be more accurate than pure semantic search. Neo4j was chosen for its expressive Cypher query language and ability to model real-world relationships in a natural wayneo4j.commedium.com.

Design considerations for Neo4j integration:

- A schema or ontology is defined (which node labels and relationship types we use). E.g., Node labels: Document, Company, Person, Issue; Relationship types: ISSUED_BY, SENT_TO, MENTIONS, etc.
- The Knowledge Graph Populator module abstracts the Neo4j queries. It will use Neo4j's Python driver (Bolt protocol) to MERGE nodes and relationships. To avoid duplication, certain unique keys are used (like a document's unique ID, or a company's name or registration number).
- The system may also use the graph to store pipeline audit info (for example, relating a Document node to a ProcessingEvent node with relationships like PROCESSED_AT timestamp). However, large logs may instead be kept outside Neo4j to keep the graph focused on domain data.

- For retrieval: The LLM Orchestrator (see below) can issue Cypher queries via an agent if needed. For example, to answer a query that involves relationships, the agent might first fetch relevant subgraph data from Neo4j, then incorporate it into the answer generation.

Both Qdrant and Neo4j are external services but **tightly integrated**. The design ensures transactional consistency to a reasonable degree: if a document is ingested, ideally both its vector and graph entries should succeed or fail together. We use the pipeline manager to coordinate this – e.g., it can use a two-phase commit simulation: first add to Qdrant, then Neo4j, if Neo4j fails, remove from Qdrant (or mark vector as orphan until reprocessing). In practice, minor inconsistencies can be corrected by a re-run (we include a reprocessing tool to rebuild vector and graph from source if needed).

3. LLM Orchestration and Agents

At the heart of Dingl AI's intelligent querying and workflow automation is the **LLM Orchestration Engine**, built using the LangGraph (or similar) framework. This component turns the system from a static data store into an interactive AI agent.

LangGraph RAG Agent: We use LangGraph to manage the conversational state and tool use for an AI agent. This agent has access to "tools" such as:

- A Vector Search Tool (which queries Qdrant for relevant text chunks given a query).
- A Graph Query Tool (which can query Neo4j for specific relationships or paths).
- Possibly a Web Search or external API tool (for future extensibility, though initial scope is internal data).
- The core **LLM (Large Language Model)** itself (which can reason and generate answers).

When a user poses a query or a task is triggered (like generating a report or detecting an anomaly), the Orchestrator engages the LLM agent. Using LangGraph, the agent is capable of multi-step reasoning:

1. It receives the user query or task request.
2. It decides (using its prompt and possibly function calling abilities) which tools to invoke. For instance, a question "Find related documents for X" might trigger a vector search for X; a question "Who approved shipment Y and when?" might trigger a graph query to find an Approval node for shipment Y.
3. The agent collects results from these tools (chunks of text, data from graph, etc.).

4. It then composes a final answer or takes an action (like raising an alert if this is part of a workflow automation).

LangGraph provides the scaffolding for this, managing the **stateful dialogue** and ensuring the agent's chain-of-thought is preservedqdrant.techqdrant.tech. The use of an agentic approach means Dingl AI can handle **complex, multi-hop queries**that require combining information across documents and databases – a significant design improvement over a simple one-shot QA system. Essentially, the LLM agent acts as a reasoning layer above our data:

- It can iterate: e.g., if initial retrieval isn't sufficient, the agent can reformulate a query (this is facilitated by giving the LLM a list of tools and allowing it to output an action like "search_vector('term Y')" which the orchestrator executes, then return the result to the LLM for further reasoning).
- It can fill gaps: e.g., if the vector search returns a snippet and the graph returns a data point, the LLM can integrate those ("Snippet says Invoice 1001 is pending. Graph says approved by John. So answer: Invoice 1001 was approved by John Doe and is pending fulfillment.").

Ollama LLM Engine: The actual LLM runs via Ollama, which allows us to host open-source models. The design abstracts the LLM such that we could use GPT-4 via API or Llama-2 via Ollama with minimal changes. For the on-prem requirement, we expect to use an open model (like Llama 2 70B fine-tuned on instructions) served through Ollama. Ollama's advantage is local inference for privacymedium.com. The pipeline manager's fine-tuning function would also leverage this: we can feed additional domain data to a fine-tuning pipeline (perhaps using LoRA or full training if feasible) and then load the updated model into the Ollama engine. This mechanism is isolated in the design so that fine-tuning does not disrupt running services (e.g., it might train in a separate environment, and once ready, the new model is swapped in).

Agent-based Modularity: Each function the agent can perform (searching vectors, querying graph, sending an alert, etc.) is implemented as a **Tool** in LangGraph's terms. This means adding a new capability, say "SummarizeDocument", is as simple as writing a function for it and registering it as a tool the agent can call. This modular approach aligns with the open/closed principle – we can extend the agent with new tools without changing the core logic. The SDD ensures that all such tools have well-defined interfaces:

- Input/Output schema: e.g., the vector search expects a query and returns a list of texts with scores.
- Execution context: the agent's state (conversation memory or query context) can be passed if needed, though often tools are stateless calls.

One important design aspect: *fail-safes and guardrails for the LLM*. We don't want the agent to hallucinate incorrect data especially in an enterprise setting. To mitigate this:

- The agent is prompted to **cite sources** or present confidence with evidence from the vector/graph data (since it has access to the chunks, it can quote or refer to them).
- Certain critical queries (like those requiring only authoritative data) might be handled by deterministic graph queries rather than LLM generation.
- We log all LLM outputs and possibly use a secondary validation (for example, run a simpler check: if the user asked for a number and the LLM gave an answer, verify that number appears in retrieved data).

4. User Interface and External Integration

The design of UI and integrations impacts how the back-end components come together for the end-user.

Dashboard (Web App): We have a web front-end (could be a React or Angular single-page app) that communicates with the backend via the API. The UI is not just a pretty layer; it also needs to reflect the system's state effectively:

- It shows a **Pipeline Configuration UI**: possibly a form or a graphical flow editor where an admin can drag-and-drop modules (OCR, Parse, etc.) into a sequence, set parameters (like OCR language, etc.), and save it. This translates to a JSON config that the Pipeline Manager uses.
- It has a **Monitoring Dashboard**: using web-sockets or periodic polling to update with pipeline run statuses. If, say, 100 documents were just uploaded, the UI can show a progress bar and any errors in real time. This might involve a push mechanism (the back-end sending events).
- **Search/Query interface:** A search bar or chat interface where a user question is sent to the LLM Orchestrator. The answer (plus maybe source highlights) is displayed. The design here considers user experience – for example, after an answer is shown, the user might click "Show sources" which would display the chunks or graph data that the agent used.
- **Graph Visualization:** For users like data analysts, we include a simple graph browser. Using Neo4j's data (via an API call), the UI can display a network graph (nodes and edges) around a particular entity. This helps trust and transparency, letting users visually inspect relationships (e.g., "show me all certificates connected to Company X" yields a subgraph display).

API Design: The REST API underlying the UI is also the integration point for external apps. For instance, a client could write a script to POST a new document to /ingest

whenever their internal system receives one. Or call /query from a chatbot in their CRM to get an answer from Dingl AI's knowledge base. To facilitate this, the API endpoints are documented with clear input/output formats. We maintain statelessness for simpler integration (the exception is the conversational query context, which if needed, we handle via a session token that the client can provide to maintain a conversation with the LLM agent across calls).

Authentication & Multi-tenancy: In the SaaS model, all API calls include a tenant ID or are on separate subdomains for each client. The architecture at the application layer ensures one client's data never mingles with another's – e.g., the vector index might prepend tenant IDs on collection names, or use entirely separate DB instances depending on scaling. On-prem, this is not a concern, and auth might integrate with local LDAP/AD.

Integration Hooks: Recognizing that Dingl AI might be part of larger workflows, we design hooks: e.g., after pipeline processing of a document is done, the system can trigger a callback or message. In the cloud, this might be implemented as a webhook: the client can configure a URL to be notified with results. In on-prem, maybe a message can be written to a local queue or file. This event-driven aspect means, for example, in Use Case 2 above, once an issue is detected, the system's alert tool could call an outgoing API (to a ticketing system) – these outgoing integrations are configured in the pipeline or rule (like a rule saying "if Rule#5 triggers, call API https://maintenance.example.com/newTicket with these parameters").

By designing the UI, API, and integration layer thoughtfully, we ensure Dingl AI is **user-friendly** and **plays well with other enterprise systems**. The UI and API don't do heavy lifting themselves; they delegate to the backend services described earlier, but they orchestrate and present results in a coherent manner.

Data Flow Diagrams

To better illustrate the interactions, we break down two primary data flows: **(A) Ingestion Flow** and **(B) Query (Retrieval) Flow**. These flows show how data moves through the components in sequence.

A. Ingestion Flow:

1. **Input Acquisition:** The user (or external trigger) initiates ingestion by providing documents (via UI upload or API). The raw files are handed to the Pipeline Manager.

2. **Preprocessing:** The Pipeline Manager invokes the OCR module for images/PDFs. For each document, if needed, OCR extracts text. In parallel (for text documents), it can skip OCR and directly go to chunking.
3. **Chunking & NLP:** The full text of each document is parsed into chunks (e.g., splitting by sections or a fixed token length). Additional NLP may run here (like identifying and tagging entities in text).
4. **Vector Embedding:** Each chunk is sent to the embedding model (via the LLM service). The returned vector and chunk metadata are saved to Qdrant in the appropriate collection.
5. **Graph Construction:** In tandem, the system extracts structured data. For example, it might detect "Invoice #123, Amount $1000, Customer ABC Corp" in the text – it then creates/updates a node for Invoice 123, links it to node "ABC Corp" with relationship "BILLED_TO," etc., in Neo4j.
6. **Rule Evaluation:** Once data from a document is embedded and graphed, the rule engine runs checks. If any rule fails (flags), those are recorded. The Monitoring module gets an event (like "Document 123 flag R1: date mismatch").
7. **Completion:** The pipeline manager marks the document as processed, and writes an entry to an internal log or status DB. This triggers an update to the UI (progress bar moves, any errors displayed). If configured, an email or webhook alert is sent for serious issues (e.g., compliance failure).

The ingestion flow is *streaming* in nature – as soon as one document is done with chunking, it can begin embedding while another document's OCR is still running. The design uses asynchronous processing and possibly a task queue (like Celery/RabbitMQ or Kafka) to manage these concurrently. This improves throughput.

B. Query/Retrieval Flow: (for an ad-hoc user question or an automated query task)

1. **User Query Input:** The user enters a question in the dashboard (or an external system calls the query API with a question).
2. **Agent Initialization:** The Query Orchestrator (LLM agent) is given the query. It has a pre-defined prompt that instructs it on available tools and how to format tool requests. For example, it might think: "To answer this, I may need to search relevant text and check relationships."
3. **Vector Search (if invoked):** The agent uses a tool to query Qdrant. This involves creating an embedding of the user question (embedding models are often part of the LLM service; we use the same embedding model as ingestion to represent the query). That query vector is sent to Qdrant, which returns the top-N similar chunkspython.langchain.com. The agent examines these chunks (they become part of its context, often appended to the prompt as information).
4. **Graph Query (if invoked):** The agent may also form a Cypher query if needed. If the question is about relationships or specific fields (e.g., "Who approved X?"),

the agent might use a template or a learned approach to query Neo4j. The results (say it returns a set of nodes/relationships or a table of values) are then fed back to the agent. We often serialize graph results in a JSON-like format for the LLM to read easily.

5. **Iterative Reasoning:** The agent now has gathered pieces of info. If it's not confident yet, it can iterate – e.g., maybe one of the retrieved chunks references another document, so the agent might do a follow-up vector search on that reference. This iteration is a loop between LLM thinking and tool usage until the agent decides it has enough information. There is usually a cap on steps (to avoid infinite loops).
6. **Answer Generation:** Finally, the agent composes the answer. It uses the LLM to generate a response in the requested format (default would be a natural language answer; sometimes it could be a table or a yes/no depending on query). Since it has context from the retrieved data, the answer is grounded in actual data, reducing hallucination. The agent might also attach references (we can design it to output which document or source was used for each part of the answer).
7. **Response Delivery:** The answer is sent back to the user interface or API caller. If it's the UI, it's displayed in the chat interface. The sources may be visualized (like highlighted text from documents). If it's via API, the JSON could include the answer text and a list of source document IDs or excerpts.
8. **Logging:** The system logs the query and answer for audit. In some cases, feedback is looped – e.g., if the user marks an answer as incorrect, that could be used to adjust future agent behavior or trigger a review of the data.

These data flows ensure that Dingl AI operates seamlessly from data ingestion to insight delivery. The design optimizes for parallelism (in ingestion) and intelligent sequential reasoning (in querying). Below is a diagram focusing on the ingestion flow (the query flow is described above textually due to complexity):

Figure: Data Ingestion Flow Diagram – Raw data flows through OCR extraction, chunking, embedding into the vector store, and relation extraction into the graph database. Monitoring and alerts are updated at each stage.

(The query flow would involve the user query hitting the LLM agent, which then interacts with the vector store and graph in a loop before producing the answer.)

Class & Module Diagrams

The implementation of Dingl AI can be further detailed with class/module diagrams, focusing on how we structure the code and components. The system is divided into packages or services, each with internal classes:

- **PipelineManager Class** (within a Pipeline Service): Coordinates tasks, holds pipeline configuration, and contains methods like runPipeline(document) which orchestrates calls to OCR, Chunker, etc. It might utilize sub-classes or handlers (Strategy pattern) for each step based on file type. E.g., OCRProcessor, TextChunker, GraphUpdaterclasses implement specific logic.
- **Extractor Module**: Classes like OCRExtractor, TextExtractor. From the user's files, the appropriate extractor is chosen (perhaps via a Factory based on MIME type). We saw hints of such design in code: e.g., a function handling .pdf, .docx, .xlsx differently.
- **VectorStore Interface**: An abstraction class VectorStoreClient which hides Qdrant specifics. It provides methods insertVectors(batch) and queryVectors(query_vector, filters) so that our core logic doesn't depend directly on Qdrant library calls everywhere. A concrete implementation QdrantClientWrapper uses the Qdrant SDK.
- **GraphStore Interface**: Similar to vector, a GraphStoreClient with methods like upsertNode(label, properties) and createRelationship(node1, relation, node2). Implemented by Neo4jClientWrapperwhich encapsulates Neo4j driver calls (like executing Cypher).
- **LLM Service classes**: Perhaps a class LLMAgentOrchestrator which uses LangGraph. Inside it, Tool classes are defined: e.g., VectorSearchTool (with an execute(query) method) uses VectorStoreClient; GraphQueryTool uses GraphStoreClient. The orchestrator has the LLM prompt template and manages calling the LLM (via an LLMClientthat abstracts Ollama vs OpenAI). If we integrate function calling, the LLM might output a JSON indicating which tool and with what params, which the orchestrator then maps to the corresponding Tool class.
- **Monitoring & Logging**: Maybe a singleton EventLogger or so, which PipelineManager uses to log events (and possibly pushes to a UI via WebSocket). This could interface with an external monitoring system (like sending metrics to Prometheus, etc., though that might be beyond scope).
- **API Controllers**: If using a web framework (Flask/FastAPI), there will be controllers for each route (IngestionController, QueryController, etc.), which translate HTTP requests into calls to the PipelineManager or LLMAgent and then format responses. These are stateless and simply coordinate with core classes.

In terms of module diagram: we essentially have **layers** – Presentation (UI & API) layer, Business Logic layer (Pipeline, Query orchestrator, etc.), Data Access layer (Vector/Graph clients), and Infrastructure (actual DB instances, LLM runtime). Each layer only interacts with adjacent layers through defined interfaces. This prevents, for example, the UI from directly calling the database without going through the logic that applies rules.

(A detailed class diagram would be inserted here, showing classes like PipelineManager, Extractors, etc., and their relationships – e.g., PipelineManager uses VectorStoreClient, which connects to Qdrant.)

【Placeholder for Class Diagram】 *Diagram would illustrate classes and relationships: PipelineManager coordinating OCRExtractor, Chunker, VectorStoreClient, GraphStoreClient; LLMAgent orchestrating VectorSearchTool and GraphQueryTool; etc.*

Deployment View

The deployment of Dingl AI in different environments can be visualized in a deployment diagram. We highlight both SaaS and On-Prem deployments:

SaaS (Cloud) Deployment:

- Dingl AI runs in a cloud environment (e.g., AWS or Azure). The components are containerized (Docker images) and orchestrated via Kubernetes for scalability. We have multiple instances of the **Pipeline Manager service**behind an API gateway or load balancer, allowing concurrent ingestion jobs.
- Qdrant is deployed as a managed cluster or a stateful set in K8s (with persistent volumes for the index data). It may scale horizontally for read loads and uses sharding for large datasets.
- Neo4j could be deployed as a causal cluster (for high availability) with a few core nodes. It also has persistent storage (SSD volumes for the graph).
- The LLM service (Ollama) is containerized possibly with GPU support. Depending on usage, we might scale it by running multiple replicas of a smaller model, or a single powerful instance for a larger model.
- The Web UI is served perhaps as a static single-page app (from an S3 bucket or via a web server container), and the API is an endpoint (maybe implemented in FastAPI/Flask) running in the same environment as Pipeline services.
- All components communicate over a virtual network. We secure it via network policies (only allow necessary traffic). For example, the API service can talk to

Qdrant and Neo4j on their ports, but no public access to those databases – public access is only via the API gateway.

- Multi-tenancy in SaaS means the **database level isolation** (could be separate DB schemas or at least tenantID filtering) and possibly separate storage buckets for any file content if stored.
- We likely integrate cloud monitoring (CloudWatch or Prometheus/Grafana) to gather performance metrics from each component. The design includes health check endpoints for each container for the orchestrator to restart if something fails.

On-Prem Deployment:

- All components reside on servers in the client's network. The typical setup might be a small cluster of 3-4 machines: e.g., one for the application (Pipeline & API), one for Qdrant, one for Neo4j, and perhaps one for the LLM if GPU hardware is separate. In a minimal install, everything could run on one beefy server for smaller scale.
- Docker Compose or Kubernetes can be used on-prem too for ease of installation. We provide a bundle that includes all services.
- The client's IT will handle exposing the UI/API internally (often via an intranet URL).
- Integration with their existing systems (like connecting to an internal SQL database for enrichment) can be configured since the system is within their LAN.
- One challenge on-prem is model updates: if we improve the LLM or add a new model, the client must install it. The SDD accounts for this by making model management a controlled process: e.g., we might package the model weights with the release or allow an offline file-based import.
- Data backups: On-prem clients need backup procedures for Qdrant indices and Neo4j data. We supply scripts or recommendations (Neo4j has backup tools, Qdrant can dump collections to files). The deployment diagram would note backup/restore components attached to those services.

Networking and Security: In both scenarios, communications are encrypted where possible. In cloud, TLS termination at the load balancer for external traffic, and internal traffic in a secure VPC. In on-prem, use of HTTPS for the web UI and API is recommended, and since it's internal, it inherits the security of the corporate network. User auth in on-prem likely ties into their directory services.

Scaling considerations:

- The Pipeline service is stateless (except for temporary job state, which can be externalized to a Redis or DB for coordination if multiple instances). So we scale it horizontally to handle higher ingestion throughput.
- Qdrant scaling might involve adding more nodes or using their cloud service if hybrid allowed. Similarly, if Neo4j becomes a bottleneck for lots of relational queries, we might direct heavy analytic queries to a read-replica.
- The LLM service scaling is interesting: if using large models, we may not easily run many in parallel due to memory constraints. We might instead queue LLM requests or use smaller models. Alternatively, the design allows using an external API (OpenAI, etc.) as a fallback in SaaS if demand spikes – but that depends on privacy needs.

【**Placeholder: Deployment Diagram**】 *Diagram would show nodes/containers: e.g., "Web Server (UI + API)" connecting to "Pipeline Service (app pods)", which in turn connects to "Vector DB (Qdrant cluster)" and "Graph DB (Neo4j cluster)". For on-prem, all within company firewall; for cloud, within a VPC with secure ingress*.

In summary, the deployment view shows that Dingl AI can be delivered flexibly: a cloud-hosted multi-tenant service or a contained on-prem solution, with the architecture remaining consistent. This was a major requirement and the design accomplishes it by abstracting dependencies and using containerization for portability.

Model Configuration & Fine-Tuning Workflows

A notable aspect of Dingl AI is the ability to fine-tune or configure the AI models to a client's needs. The SDD outlines how this is done in practice:

Model Configuration: This refers to selecting which pre-trained models to use for various tasks. For example:

- OCR: We may allow using Tesseract OCR vs. an ML-based OCR (perhaps a CNN model for handwriting, etc.). The configuration would be in a settings file or admin UI (e.g., "OCR Engine = Tesseract" or path to a model).
- Embedding Model: By default we might use an open-source model (like SBERT or Instructor XL). If a client has specific needs (e.g. multilingual support), they might configure a different embedding model. Because we use the VectorStoreClient abstraction, as long as the embedding yields a vector of a known dimension, Qdrant usage remains the same. The system might load the embedding model at startup of the LLM service (Ollama could handle this if the embedding model is implemented as an LLM or we use a Python model).

- LLM for QA: The Ollama service can host multiple models. Configuration might include which model to use for answering questions. For instance, "use Model A (7B parameters) for general queries, but Model B (fine-tuned, 13B) for compliance-related questions" if needed. Possibly we let advanced users define that certain queries route to certain models (this could be achieved by the orchestrator if we implement such logic, though initial scope might be one main model).

The configuration data could be stored in a YAML or in the database (especially if different per tenant). The system reads these settings upon initialization of components.

Fine-Tuning Workflow: If a client has proprietary data and jargon that the base model doesn't handle well, they may want to fine-tune the LLM. Our design allows two kinds of fine-tuning:

1. **Embedding model fine-tuning** – less common, but if needed, they could fine-tune embeddings to better cluster their domain text.
2. **LLM fine-tuning** – to improve answer accuracy/style on domain-specific questions.

We assume the use of **LoRA (Low-Rank Adaptation)** or similar techniques to fine-tune large models efficiently. The workflow:

- The user (with admin rights) goes to a Fine-tuning section on the dashboard. They upload a fine-tuning dataset: for example, a set of sample question-answer pairs or documents with desired outputs.
- The system verifies and formats this data. It might use a script to convert Q&A pairs into a training format (maybe a JSONL that the LLM training script expects).
- Dingl AI either connects to an internal training pipeline or exports the data for offline training. If we include training capability, and if the server has a GPU, we could launch a training job using something like Hugging Face Transformers or an Ollama fine-tune command if supported.
- During training, the user might see progress (we could integrate something like TensorBoard or just log updates).
- Once training is done, a new model artifact (e.g., a diff or LoRA weights) is produced. The system then **deploys** this model: in Ollama, one might import the model or run an ollama push command to make it available as an endpoint.
- The configuration is updated so that the LLM Orchestrator now uses the fine-tuned model for queries. We keep the original model available too in case of rollback or for other general queries if needed.

- We record the version of the model and possibly evaluate it: the system could run some test queries (maybe provided by user or basic ones) to ensure it's responding sensibly.

It's important that fine-tuning is done safely (especially if using a client's GPU machine – we should ensure it doesn't monopolize resources). The design could schedule fine-tuning jobs in off-hours or throttle usage.

For models like Neo4j and Qdrant, "fine-tuning" doesn't apply, but **configuration** does: e.g., you might adjust Neo4j memory settings or Qdrant distance metric (cosine vs Euclidean). Those are lower-level configs likely not exposed to end users, but we as developers set them appropriately (cosine similarity for embeddings typically).

Continuous Learning: Beyond one-off fine-tune, the design can support continuous improvement. For instance, if users correct answers or highlight relevant passages, we could feed this back to retrain the model or adjust vector weights (some active learning). This is an aspirational part of design: initial version might log user feedback and require manual retraining triggers.

Model Versioning: Each model and its fine-tuned variants should be versioned. The configuration might include model name and a version hash. The system could allow rolling back to a previous model if the new fine-tune performs worse. This implies storing multiple models and switching – which is doable as long as we have disk space and manage which one is loaded by the LLM service.

Testing and Validation: After fine-tuning or model config changes, the system can run a set of **unit tests** (some sample queries that we expect known outputs for). The results can be shown to the user (e.g., "Baseline model answer vs Fine-tuned model answer" for some queries, illustrating improvement). This helps justify the fine-tuning and catch any regressions (like new model may have gained domain knowledge but maybe got slower or lost some general capability – something to watch).

Given the criticality in enterprise, not all clients will fine-tune models themselves. They may ask us (the provider) to do it or they may rely on few-shot prompting. So our design must also accommodate an alternative: **Prompt Customization**. Instead of full fine-tune, an admin might provide custom system prompts or examples to the LLM agent for certain query types. The SDD can treat this as configuration (like a YAML with prompt templates per scenario, which the orchestrator reads and applies). This is a lighter-weight way to adjust behavior. For example, a client might want the tone of answers to be very formal, or always include source citations – we can achieve that via prompt engineering, configurable per tenant.

In summary, the model configuration and fine-tuning workflow in Dingl AI is designed to be as user-friendly and safe as possible, given the complexity:

- Clear steps and UI for providing training data.
- Using efficient fine-tuning methods (to not require enormous compute time).
- Mechanisms to deploy the new model and fall back if needed.
- Maintaining privacy (the client's data used for fine-tune stays on-prem or in their cloud instance; if we assist with fine-tune in SaaS, we will isolate their data and model).

This flexible model management differentiates Dingl AI in being not just a static product but an evolving one that can learn the specifics of each enterprise's data over time, thereby increasing accuracy and value.

(A sequence diagram could be added here to illustrate the fine-tuning process: Admin -> uploads data -> "FineTuneService" -> training -> new model -> updated inference path.)

Dingl AI – Business & Technical Whitepaper

Executive Summary

Problem: Modern enterprises grapple with an **overwhelming volume of unstructured data** – documents, images, and even videos – scattered across departments. Extracting meaningful information from these sources is often a manual, error-prone process. Critical business workflows (like compliance checks, approvals, and audits) suffer from delays and inconsistencies because data isn't readily accessible or cross-linked. This leads to inefficiencies and missed opportunities in an era where timely insights are key.

Solution: Dingl AI is an AI-powered platform that addresses this challenge by **ingesting multimodal enterprise data and transforming it into an intelligent knowledge hub.** It combines cutting-edge document and process intelligence with workflow orchestration to automate data extraction, link related information, and provide instant answers to user queries. Powered by advanced large language models (LLMs) and a hybrid vector database + graph database approach, Dingl AI can, for example, parse hundreds of supply chain certificates in minutes, flag compliance issues, and answer questions like *"Which shipments are missing approval documents?"* on the fly.

How it Works: Dingl AI's pipeline ingests documents (text, PDFs, images, video transcripts), applies OCR and natural language processing to extract content, then stores knowledge in a semantic vector store and a relational graph store. This enables both fuzzy semantic searches and exact relationship queries. An integrated AI assistant (LLM agent) orchestrates these components to answer natural language questions or trigger automated workflows. Users can configure pipelines to suit their processes and even fine-tune the AI on their proprietary data for better accuracy. Real-time dashboards and alerts provide visibility and oversight, crucial for high-compliance industries.

Value Proposition: By automating data ingestion and making enterprise knowledge instantly accessible, Dingl AI dramatically **reduces manual effort** (IDP technologies can cut document processing work by up to 80%straitsresearch.com) and speeds up decision-making. It ensures **traceability**, meaning every piece of data is linked and auditable, which is vital for compliance and quality control. Early adopters can expect not only labor savings but also improved data quality (fewer errors), faster cycle times for workflows, and enhanced ability to derive insights (via unified search and AI Q&A).

Market Position: Dingl AI stands at the intersection of intelligent document processing, RPA (robotic process automation), and the emerging field of AI orchestration. Unlike traditional RPA or OCR solutions, Dingl AI offers a **unified, AI-first approach** that learns from data and becomes smarter over time. It's especially valuable for B2B firms in sectors like manufacturing, supply chain, energy, and finance where document trails and field data collection are heavy. With flexible deployment (cloud or on-prem) and enterprise-grade security, Dingl AI is poised to become the go-to platform for companies looking to leverage AI on their operational data without sacrificing control or compliance.

In summary, Dingl AI enables enterprises to unlock the value hidden in their documents and processes by converting unstructured data into structured intelligence and actionable workflows. It offers an immediate efficiency boost and a strategic advantage in harnessing organizational knowledge, all while fitting into existing IT ecosystems.

Problem & Solution Overview

The Problem: Enterprises today face an **information deluge**. Critical information is trapped in diverse formats: invoices and contracts in PDF, inspection photos and videos, emails, and legacy reports. Manually processing these is slow and expensive. Important processes – like verifying a supplier's certificates or approving a field service report – can stall as employees search for data or re-key information from one system to another. This not only wastes time but introduces errors and compliance risks. For

example, in supply chain audits, missing or mismatched documents can lead to costly delays or regulatory penalties. In many organizations, knowledge is siloed; one department's documents are not easily cross-referenced to another's, obscuring insights (e.g., linking a customer's contract to their support tickets might reveal upsell opportunities, but only if those data sources "talk" to each other).

Traditional solutions address fragments of this problem: OCR tools digitize text but don't understand context; BPM/workflow software routes tasks but doesn't intelligently read content; and general BI tools struggle with unstructured data. There is a pressing need for a solution that **combines data understanding with process automation**.

The Solution – Dingl AI: Dingl AI tackles these challenges with an integrated platform that offers:

- **Automated Data Ingestion:** It can plug into existing data streams (scanners, emails, data exports) and automatically pull in documents, images, and other media. Using AI, it **extracts text and key fields** from these sources far more accurately than manual data entry. For instance, Dingl AI can read an equipment maintenance log (even if handwritten or scanned) and capture the critical details.
- **Unified Knowledge Base:** Dingl AI creates a **knowledge graph + vector index** of all ingested content. This means all data becomes queryable. Unlike a simple database, it keeps the context – not just "what is in a document" but "how documents relate to each other." For example, a purchase order, its invoice, and the shipping document can be linked by order number or entities. This solves the data silo issue by overlaying a connective tissue across documents.
- **Intelligent Process Orchestration:** Beyond static data, Dingl AI forms the backbone of dynamic workflows. It can be configured to trigger actions or notifications based on content. If a certain field is missing in a document, it can automatically request it; if a condition is met (say a safety report mentions a critical fault), it can alert a supervisor immediately. In effect, Dingl AI doesn't just surface information – it helps drive processes forward with that information.
- **AI-Powered Query & Insight:** The platform's conversational AI interface is a game-changer. Users can ask questions in plain English (or other languages) and get answers synthesized from across all their data. This is **solution-oriented AI** – the user doesn't need to know where the information resides or how to retrieve it; Dingl AI figures that out. It's like having an expert analyst available 24/7 who has read all company documents and can summarize or answer anything about them. This addresses the knowledge retrieval problem in a way traditional search cannot, because the AI understands context and intent, not just keywords.

- **Fine-Tuning & Adaptability:** Recognizing that each enterprise has unique terminology and needs, Dingl AI allows customization. The AI models can be fine-tuned on company data and jargon, improving accuracy over time. Pipelines can be tailored – e.g., a bank can add a custom step to check a document against sanction lists, or a manufacturer can integrate IoT sensor data into the analysis. The solution is not one-size-fits-all; it's a flexible foundation that adapts to specific problem statements.

Example Scenario (Problem vs. Dingl AI): Consider a compliance manager trying to verify that every shipment has a corresponding certificate of origin and lab test report. Historically, they would manually cross-check spreadsheets and PDFs, often across different systems – a tedious and error-prone task taking days. With Dingl AI, the ingestion pipeline automatically links shipments to their documents. The manager can simply ask, *"Are any Q3 shipments missing required certificates?"* Dingl AI's agent checks the graph, finds the mismatches, and responds with a list of shipments (and even highlights which document is missing). Moreover, it could automatically notify the respective site manager about the missing document – **problem solved proactively**.

In summary, Dingl AI provides a holistic solution that spans **data capture, understanding, and action**. It directly addresses the pain of manual data processing (saving time and reducing errors), breaks down information silos (by creating a connected knowledge base), and infuses intelligence into processes (via AI-driven Q&A and automation). The result is an enterprise that operates with far greater efficiency and insight:

- Faster processing cycles (weeks into hours),
- Enhanced decision-making (since all relevant info is at one's fingertips),
- Better compliance and traceability (everything is logged and linkable),
- Freed-up human talent (teams can focus on analysis and exceptions, not rote data handling).

Dingl AI's approach epitomizes the **problem-solution fit** for organizations drowning in unstructured data but hungry for structured actions.

Competitive Advantage

The landscape of enterprise automation and AI has several players – from legacy OCR vendors to RPA (Robotic Process Automation) platforms and emerging AI startups. However, Dingl AI offers a **unique combination of capabilities** that sets it apart:

- **Multimodal & Multilingual Mastery:** Many competitors offer document processing but are limited to text or specific formats. Dingl AI natively handles

text, images, and video transcripts under one roof. For example, a competitor might require separate tools for invoice OCR and for analyzing a training video transcript, whereas Dingl AI can ingest both into the same knowledge graph. This multimodal flexibility means Dingl AI can be deployed in varied scenarios (from scanning paper archives to analyzing bodycam footage for field operations) without needing a patchwork of tools. Fewer systems and integrations mean lower total cost and higher consistency.

- **Integrated Vector and Graph Approach:** Typical IDP (Intelligent Document Processing) solutions put extracted text into a search index or database, but they often miss the relationships between data points. Dingl AI's **dual storage** (vector for semantic similarity, graph for explicit linkage) is a game-changer. It can answer fuzzy questions (thanks to vectors) and also enforce precise constraints (thanks to the graph). Most competitors choose one or the other. This gives Dingl AI a **technical edge** – it can, for instance, find "documents similar to this contract" and also trace "who signed this contract and what else did they sign" in one system.
- **LLM-Powered Agent**: While some enterprise tools are beginning to integrate AI, Dingl AI was built around an **LLM agent from the ground up**. This means it doesn't just do what you program it to do; it can handle novel queries and tasks using AI reasoning. Competing platforms often require pre-defined workflows or queries – if you ask something slightly outside those, they falter. Dingl AI's agent can dynamically figure out how to answer a new question by juggling its toolsqdrant.tech. It's akin to the difference between a hard-coded chatbot and ChatGPT – the latter can handle an open domain. In enterprise context, this flexibility is gold because business questions evolve constantly.
- **User-Defined Pipelines & Extensibility:** Many big-name solutions (IBM, ABBYY, etc.) offer IDP but in a black-box or rigid way. Dingl AI prides itself on **user-defined pipeline configuration**. The ability for users to tailor the sequence of processing steps (and even insert custom logic or connectors) is a massive advantage in B2B scenarios where one size doesn't fit all. This configurability is achieved through a simple interface, meaning you don't have to write code to change how Dingl AI works. Competitive products might require costly custom development to achieve what Dingl AI can do through configuration. This lowers barrier to adoption and tweaks.
- **Fine-Tuning and Learning:** Dingl AI can **learn from your data**. If a competitor's product has an out-of-the-box model, you're stuck with its accuracy limits. Dingl AI allows fine-tuning the underlying models on a client's specific documents – improving over time. Essentially, the more you use Dingl AI, the smarter it can get for your domain. This ongoing improvement is a competitive moat;

customers aren't just buying static software, they're investing in a growing asset tailored to them.

- **Deployment Flexibility (Cloud & On-Prem)**: Some emerging AI companies offer great cloud services but no on-prem option, which cuts off clients in regulated industries. Conversely, old guard enterprise software might offer on-prem but lacks a modern SaaS or is not truly cloud-native. Dingl AI provides both: a secure multi-tenant cloud for those who want hassle-free usage, and a self-hosted version for those who need data on their own servers. This **dual model** means we can serve a wider market. Moreover, the on-prem is not a reduced "lite" version – it's full-featured, which is a strong selling point against competitors who might compromise features for on-prem due to dependencies on cloud.
- **Comprehensive Workflow Orchestration:** RPA tools like UiPath or Automation Anywhere are great at automating screen clicks or routine tasks, but they don't deeply understand content. Dingl AI, with its process orchestration, can handle content-based decisions (like "if the document says hazardous, trigger a special review"). We essentially bridge the gap between RPA and IDP with cognitive intelligence. This stands as a **competitive advantage** where enterprises currently might use three different products (OCR, RPA, BI) to achieve what Dingl AI does in one integrated flow.
- **Ease of Use and Quick ROI:** We focus on a clean UI and quick deployment. For example, Dingl AI comes with templates for common use-cases (invoice processing, contract analysis, etc.), so clients can get started in days, not months. Competing enterprise solutions often have long implementation cycles. A quicker time-to-value is a big advantage in the market – companies can run a pilot with Dingl AI and see tangible results in a single quarter (like reducing document processing time by 50% or cutting errors significantly). This contrasts with traditional IT projects that might take a year before delivering. Our early pilot references will demonstrate such quick wins.
- **Cost-Effectiveness:** Because Dingl AI leverages open-source components (like Qdrant, Neo4j community, and open LLMs via Ollama) and commodity hardware (can run on standard servers), our cost base can be lower. We can pass this savings in pricing (see Pricing Strategy). Many established players have high licensing costs or require proprietary hardware/appliances. Dingl AI can often be introduced using existing infrastructure (especially on-prem, e.g., run on existing VMware or in cloud subscriptions the client already has) – a subtle but important advantage in cost-sensitive markets.

Why Competitors Can't Easily Replicate This: A key competitive moat is the **technical integration** we've achieved. It's non-trivial to build a system that seamlessly marries LLM reasoning with live data stores; we have a first-mover advantage in that space focusing on enterprise data. Larger competitors may have components but

integrating them (e.g., an OCR module talking to a knowledge graph and then to a language model) involves overcoming organizational silos and legacy architecture – not an overnight task. Smaller startups might have cool AI but lack enterprise features like on-prem or robust security/compliance measures that we bake in. Dingl AI's team and product design cross these domains (AI, databases, enterprise integration), which is a rare combination and thus a strong competitive differentiator.

In summary, Dingl AI's competitive advantage lies in being a **holistic, intelligent platform** that is flexible, learning, and user-friendly. It offers capabilities under one umbrella that would typically require a mashup of multiple tools. This not only reduces cost and complexity for clients, but also unlocks synergies between data and processes that siloed tools cannot achieve. It positions Dingl AI not just as a tool, but as an AI partner embedded in the enterprise's knowledge ecosystem – a value proposition that is hard to match.

Market Opportunity

The market for AI-driven enterprise solutions is experiencing explosive growth. Dingl AI sits at the convergence of several high-growth markets: **Intelligent Document Processing (IDP)**, **Intelligent Process Automation**, and **Enterprise AI/Knowledge Management**. Let's quantify the opportunity:

- The **global Intelligent Document Processing market** was valued around $2.4 billion in 2024 and is projected to skyrocket to over $37 billion by 2033straitsresearch.com, at a CAGR of 35%+. This reflects the urgent push by companies to automate document-heavy workflows and the increasing volumes of unstructured data.
- The broader **Intelligent Process Automation** market (covering automated workflows, RPA with AI, etc.) was about $14.5 billion in 2024 and is forecast to reach $44.7 billion by 2030grandviewresearch.com at ~22.6% CAGR. This indicates strong demand for technologies that improve process efficiency and reduce human intervention in routine tasks.
- The **Enterprise AI** market (which includes NLP, knowledge graphs, etc.) is also expanding rapidly as organizations invest in AI to stay competitive.

Figure: Intelligent Process Automation Market Size Growth – Illustrating growth from $14.6B in 2024 to $44.7B by 2030grandviewresearch.com.

Key Drivers of Market Growth:

1. **Data Explosion and Digital Transformation:** Companies are drowning in data. Over 80% of enterprise data is unstructured (documents, emails, media). There's a huge opportunity for solutions that can leverage this untapped data. Digital transformation initiatives, accelerated by remote work and need for efficiency, allocate budgets specifically for automation and AI to handle data.
2. **Labor Costs and Skill Gaps:** Manual processing of documents is not only slow, it's costly and subject to human error. With rising labor costs and a shortage of skilled analysts, businesses are keen to deploy AI to do the heavy lifting, freeing humans for higher-value work. The ROI from reducing manual effort (like the 80% efficiency gains from IDP noted in some casesstraitsresearch.com) is very attractive.
3. **Regulatory Compliance and Risk Management:** Industries such as finance, healthcare, and supply chain are under growing regulatory scrutiny requiring thorough documentation and audit trails. The market is seeking tools that improve compliance – Dingl AI's traceability and automated checks speak directly to this need. For example, new data privacy laws and ESG reporting requirements compel companies to systematically manage documents and data, driving demand for robust solutions.
4. **AI Maturity and Acceptance:** A few years ago, using AI for core business processes was experimental; now it's becoming mainstream. The success of LLMs like GPT has opened executives' eyes to what's possible with AI beyond chatbots – such as understanding and generating business content. There is a window of opportunity right now where businesses are allocating budget for "AI projects" – and Dingl AI can capture those by clearly demonstrating value on core operations.
5. **Competitive Pressure:** Early adopters in various sectors (banks automating loan processing, logistics firms automating paperwork, etc.) have started pulling ahead. This is pressuring others to invest in similar technology to not fall behind. Thus, the TAM (Total Addressable Market) is not limited to one sector – any mid-to-large enterprise with document-heavy workflows is a potential customer.

TAM, SAM, SOM Analysis:

- **TAM** (Total Addressable Market): Considering all industries worldwide that could use document & process AI, TAM is in the tens of billions (summing IDP, RPA, AI software markets). For instance, if IDP hits ~$37B by 2033 globallystraitsresearch.com and IPA in mid-$40B by 2030, TAM for Dingl AI (sitting in both) could conservatively be around $50B+ by 2030.
- **SAM** (Serviceable Available Market): We focus initially on particular verticals: manufacturing and supply chain (for traceability), financial services (for document processing and compliance), and energy/utilities (for field data).

Geographically, initial focus is domestic (we can expand to US/Europe next). The SAM over the next 5 years might be, say, 20% of TAM – so on the order of $8–10B.

- **SOM** (Serviceable Obtainable Market): As a startup/new product, capturing even a single percentage of SAM is huge. In the next 3 years, if we aim for ~0.1% of TAM, that's ~$30–50M revenue potential, which aligns with capturing a few hundred enterprise customers globally (which is feasible with B2B sales focusing on high-value clients).

Market Segmentation: We see high readiness in:

- **Supply Chain & Manufacturing:** Need to track certificates, compliance docs, quality reports. Many still use manual processes – ripe for us.
- **Financial Services:** Banks, insurance – dealing with loan docs, claims, KYC paperwork. Already spending on AI, easier to sell value of improved processing speed and compliance.
- **Energy/Oil & Gas/Utilities:** Lots of field inspection reports, safety documents, engineering drawings, etc. and strong compliance needs.
- **Healthcare (Secondary focus):** Hospitals and pharma have intense documentation (patient records, regulatory submissions). It's a big market but requires some domain specialization; still, our tech is applicable.
- **Government/Education:** Also heavy on documents, but sales cycles are slower here.

The market is also expanding with the concept of **AI Orchestration** – coordinating multiple AI and data pieces to deliver outcomes. According to some reports, the AI Orchestration market was valued at ~$7.5B in 2023 and might reach ~$43B by 2032snsinsider.com, reflecting ~21% CAGR. Dingl AI exactly fits that narrative, as it orchestrates data ingestion and AI reasoning.

Figure: Projected Deployment Mix of Dingl AI by clients – indicating about 65% opting for cloud SaaS vs 35% on-prem in early years, reflecting cloud adoption trends.

Target Market Adoption and Trends:
Initially, we expect tech-forward organizations (with Chief Digital Officers, innovation budgets) to pilot Dingl AI. These early adopters set the stage, and as we prove ROI, we move into the early majority. Cloud adoption trends mean many will accept a SaaS solution; however, a sizeable segment (esp. in manufacturing and finance) still prefer on-prem or private cloud, hence our offering mix. As seen in the figure above, we anticipate perhaps a 65/35 split between SaaS and on-prem clients in the near term, shifting more towards SaaS as trust in cloud grows.

Another trend is the move from point solutions to platforms. Companies are tired of juggling dozens of software tools. Dingl AI's all-in-one approach aligns well with a market desire for **consolidation** – one platform to handle what used to require many.

Competitive Landscape (Market perspective): There are established players (e.g., ABBYY in IDP, UiPath in RPA, IBM/Google with point AI solutions) and a flurry of startups. The market isn't waiting around – it's moving now and expected to have multiple winners. But given the growth rates, there's room for new entrants that differentiate. Key is to carve out our niche: end-to-end multimodal workflow intelligence. We're not selling just OCR or just a chatbot; we're selling operational transformation powered by AI. That message resonates with the current boardroom priority of AI adoption.

In summary, the market opportunity for Dingl AI is vast and growing. By 2025–2030, enterprises that haven't embraced intelligent document and process automation will be at a severe disadvantage. Dingl AI is well-positioned to capture this wave with a solution that meets an acute need. Our go-to-market will focus on those high-value sectors and use-cases where we can quickly prove impact, then broaden. Given the size of budgets being allocated to AI in the enterprise (Gartner predicts something like 70% of organizations will adopt AI-based automation by 2025), our timing is excellent. If we execute well, Dingl AI can ride these tailwinds to substantial scale and become a key player in the transformation of how businesses handle their critical data and processes.

Pricing Strategy

Dingl AI's pricing strategy is designed to be **transparent, scalable, and aligned with value delivered**. We have a **tiered pricing model** combined with usage-based components, ensuring clients of all sizes can adopt the platform and grow with it. The tiers (Basic, Pro, Enterprise) differ by features and capacity, while usage-based billing ensures fairness for the actual amount of processing done.

Pricing Tiers:

- **Basic:** Targeted at small teams or as an entry-level for a single department use. This is offered primarily as a **SaaS subscription**. It includes core features: ingestion of common document types, the AI Q&A interface, and basic dashboarding. It might limit the volume of documents per month (e.g., up to 5,000 pages/month) and possibly the complexity (maybe fewer custom pipeline tweaks). Basic tier provides only cloud deployment (no on-prem) and standard support (community/forum and email with 2-business-day response). **Price**: for

example, $X per month (could be around $5k/month range) with an overage charge if pages exceed limit.

- **Pro:** Suited for mid-sized enterprises or larger deployments within a company. Pro unlocks higher usage (say 50,000 pages/month included), more advanced features like custom pipeline configurations, integration APIs, and perhaps one custom fine-tuned model included. It offers priority support (e.g., 24h response SLA) and an onboarding package. Pro tier could be available both in SaaS and as a single-node on-prem install (for those who require data on-site but not high availability clustering). **Price**: maybe $Y per month (e.g., $15k–20k/month) plus usage.
- **Enterprise:** This tier is for large organizations with heavy usage and advanced needs (multiple departments, high compliance requirements). Enterprise includes **unlimited or very high volume** document processing (with usage-based billing beyond a base), all features (full customization, ability to fine-tune multiple models, on-premise cluster deployment or VPC deployment in cloud, etc.). It also includes premium support (dedicated account manager, 99.9% uptime SLA, on-call support). Pricing here is typically custom – likely an annual contract in the six to seven figures depending on scale. We might start with e.g., $Z per year covering X million pages and then a per-page fee beyond. Enterprise also allows source code escrow or specific contract terms if needed and can include professional services (like we help set up pipelines, do custom dev) either bundled or at an additional cost.

Usage-Based Component: We meter certain aspects:

- **Document/Page Count:** As above, beyond a threshold, there's a per-page fee (e.g., $0.01 per page processed or similar). This aligns cost to actual usage – if a client suddenly processes double the docs, our costs (compute) go up, so usage fees cover that. It also means smaller clients pay less, larger ones pay more, fairly.
- **API Calls/Queries:** We might also meter API calls or AI query usage especially for SaaS (like how some services charge per 1,000 queries). For example, Basic might include 1,000 AI queries/month and then $X per additional 100 queries. This ensures heavy use of the LLM (which has significant compute cost) is covered.
- **Additional Modules:** Fine-tuning a model could be an add-on fee (since it uses a lot of compute and is a one-time heavy task). Or advanced modules like a very high-res OCR for handwriting might consume more resources, so either locked to higher tier or usage-billed.

Pricing Rationale:

- **Value-based:** Dingl AI is delivering potentially massive savings (in labor hours, error reduction). Our pricing captures a fraction of that value. For example, if we save a company 5 FTEs worth of manual work, that's say $300k/year – paying us $100k/year is an easy yes. We will price in a way that demonstrates ROI: e.g., aim for 3-5x return for the customer.
- **Competitive Positioning:** We'll research competitor pricing (often enterprise software is not public, but we know RPA bots can be ~$10k each annually, IDP engines can be ~$0.05/page, etc.). We position ourselves maybe a bit below the established players to lower barrier (since we're newer), but not too low to avoid being seen as low-quality. For instance, if a competitor charges $0.10 per page processed, we might aim for effective $0.05-$0.08 range depending on volume.
- **SaaS vs On-Prem:** SaaS subscriptions cover hosting costs so they have usage baked in. On-Prem customers often expect licensing based on usage or server size. We might license per server or core for on-prem if unlimited usage on that server. Alternatively, an on-prem Enterprise could be site-license unlimited usage at a fixed price (which could be high but gives them predictability). We can remain flexible.
- **Scaling with Customer Growth:** As customers expand usage, they can smoothly move from Basic to Pro to Enterprise or just pay the overages. We want to avoid surprising bills, so we provide clear monitoring of usage and maybe volume discounts at higher usage (for example, the per-page cost could drop once they pass a certain volume per month).
- **Trial and Freemium considerations:** Perhaps offer a limited free trial (e.g., process 500 pages free or a 1-month free pilot with limited features) to reduce risk for new customers. This is common in SaaS and helps acquire leads. For on-prem, a POC free period can be allowed with a time-locked license.

Example Pricing Card Representation:

Basic	Pro	Enterprise
Starting at $5,000/mo	Starting at $15,000/mo	Custom (> $100k/yr)
Up to 5k pages/mo included	Up to 50k pages/mo included	Volume negotiated (millions)
AI Q&A (500 queries/mo)	AI Q&A (5k queries/mo)	Unlimited queries
Standard support	Priority support (24h)	24/7 dedicated support
Cloud only	Cloud or On-Prem (single server)	Cloud VPC or On-Prem cluster
–	Fine-tune 1 model included	Fine-tune multiple models
–	–	Custom integrations & SLAs

(Exact numbers TBD, but this conveys structure.)

【**Pricing Cards Placeholder**】 *Our marketing site will show pricing tiers with feature checklists for clarity (e.g., a table or card layout highlighting differences).*

The key is that our pricing should be **predictable for budgeting** yet **flexible for usage spikes**. We achieve predictability with the tier base fees and flexibility with usage billing. Over time, as we optimize costs (e.g., running models more efficiently), our margins improve, but initially pricing also factors in cloud compute costs – e.g., heavy LLM usage by a client will be offset by the usage charges.

We'll also consider **discounts for annual commitments** (e.g., 2 months free on annual pre-pay) to encourage long-term engagement, and volume discounts for multi-year or multi-entity deals (if a conglomerate wants to deploy to 5 subsidiaries, we'll do a group deal).

In conclusion, our pricing strategy aims to lower entry barriers (a department can start in Basic tier relatively cheaply), then expand and capture more value as the customer sees results and rolls Dingl AI out more broadly (scaling into Pro/Enterprise and higher usage). By aligning cost with value (through usage metrics that correlate with documents processed and insights generated), we make a fair exchange that can be justified easily in procurement processes. We will continuously revisit pricing as we learn usage patterns and as our costs (especially related to LLM serving) evolve.

Deployment Models

Dingl AI offers flexible deployment models to meet the needs of different customers: **Cloud (SaaS)**, **On-Premises**, and even hybrid variations. This flexibility is a key selling point given varying IT policies and is structured as follows:

- **SaaS (Cloud) Deployment:** In the cloud model, Dingl AI is hosted by us (the provider) in a secure cloud environment (e.g., AWS or Azure). This is a multi-tenant setup where infrastructure is shared, but each customer's data is isolated logically. The SaaS model is ideal for customers who want a hassle-free experience – no installation, immediate updates, and the ability to scale usage on demand. We handle all maintenance (backups, uptime, security patches). From the customer's perspective, they just access the platform via a web browser and API endpoints. SaaS is also beneficial for rapid innovation – we can roll out new features continuously. We ensure the SaaS meets enterprise security standards: encryption at rest and in transit, ISO27001 compliance for our cloud operations, etc. Likely, a majority of new customers (especially mid-market companies) will opt for SaaS to minimize their DevOps overhead.

- **On-Premises Deployment:** For clients with strict data residency, security, or compliance requirements (common in finance, government, defense, certain manufacturing), we offer an on-premises deployment. This means Dingl AI's components are installed in the customer's own data center or private cloud. It can be delivered as a set of Docker containers or VMs along with installation scripts. The on-prem version has the same core functionality, but gives the client full control over data and access. No sensitive data leaves their environment. We typically recommend a dedicated server or cluster for Dingl AI depending on scale (with appropriate CPUs, memory, and optionally GPUs for the AI). On-prem deployment may require more collaboration with the client's IT – setting up prerequisites (e.g., ensuring they have Docker and required OS), configuring integration points (like connecting to their Single Sign-On system or databases if needed). We provide documentation and support for this. Updates on-prem are provided as periodic releases (e.g., every quarter, a new version that they can choose to upgrade). The on-prem model often goes hand-in-hand with the Enterprise tier pricing and possibly a different licensing mechanism (e.g., license keys tied to servers).
- **Hybrid Deployment:** Some customers might choose a hybrid approach – for example, keep their data storage on-prem but use our cloud for the heavy compute, or vice versa. While not the primary offering, our architecture can accommodate it. For instance, a customer could run the vector and graph databases on-prem (so all data stays in-house), but use our cloud LLM service to process queries, with only embeddings (not raw sensitive text) going to cloud. Or a scenario: initial data ingestion on-prem, then pushing anonymized or summarized data to cloud for broader analysis. We can support such custom deployments for clients who request it, ensuring that any sensitive pieces remain where they're comfortable. This hybrid model might be relevant if a client has, say, a secure network for data but wants to leverage cloud GPU power for the AI – we'd establish secure VPN or dedicated connections for the communication in that case.

Deployment Mix and Strategy: Based on market trends, we anticipate perhaps around 60-70% of new customers will choose our SaaS, and 30-40% will go on-prem (as represented earlier in the donut chart). Over time, as trust in cloud increases in laggard industries, the SaaS portion could grow. However, our strategy is to not leave any segment untapped – by offering on-prem, we gain access to clients that our cloud-only competitors might lose. It does mean extra work (maintaining deployment flexibility), but it's justified by the additional revenue streams.

Technical Considerations per Model:

- In SaaS, we invest in robust multi-tenant security. Each tenant's data separated at the database level, and the AI models handle context per tenant. We likely have separate indexes/graph DB namespaces per tenant to avoid any possibility of data bleed. We also consider performance isolation – heavy usage by one client shouldn't degrade others (this can be handled via auto-scaling and maybe throttling per tenant if needed).
- In On-Prem, we ensure our software can run in environments without internet access (often required). That means packaging models and dependencies offline. Licensing for on-prem might be enforced via license servers or simple trust model with audits. Also, support is a bigger factor – on-prem clients might need more hand-holding on upgrades and troubleshooting since we can't directly see their system easily.
- Compliance: Some customers might require that even SaaS be deployed in a specific region (for data residency). We can accommodate with region-specific hosting (e.g., have an EU deployment for EU customers, etc., or offer a private single-tenant cloud deployment in their Azure/AWS account – which is another flavor of deployment, essentially on-prem in their cloud).
- Customization: Enterprise clients often want custom integrations or slight tweaks. On SaaS, we maintain a common codebase (we'd implement features that benefit multiple customers rather than one-off custom code). On on-prem, sometimes you can inject custom scripts or connectors (which our pipeline configuration allows to some extent). We handle this via plugin interfaces or providing professional services to tailor as needed.

Transitioning Deployment: We also consider that a customer might start in SaaS (pilot, smaller scale) and then decide to move on-prem (or vice versa). We will facilitate data export/import and setup to **avoid lock-in**. A smooth path between deployment models is itself a selling point (we can say: try on cloud, if you later need on-prem, we can shift you easily).

In conclusion, our multi-modal deployment strategy ensures we can serve the broad enterprise market:

- **SaaS** for fast onboarding and broad reach (particularly mid-market and any who are cloud-friendly).
- **On-Prem** for high-compliance clients (banks, etc.) and those with large existing infrastructure investments.
- **Hybrid/Custom** for those with specific architecture needs, showing our flexibility and customer-centric approach.

This approach not only maximizes market coverage but also future-proofs us. For example, if sovereignty issues become big (certain countries mandating local data only), we can deploy in those jurisdictions (either via on-prem or local cloud region) rather than losing those opportunities.

From a business perspective, SaaS yields recurring revenue and easier scalability; on-prem often yields larger upfront license fees and commitment. By having both, we balance our revenue streams and cash flow. Over time, we expect subscription (whether cloud or on-prem subscription) to be the model, moving away from one-time licenses – ensuring we have continuous relationships and revenue.

Financial Roadmap

Our financial roadmap outlines how we plan to grow revenue, manage expenses, and achieve profitability over the next several years, given our go-to-market strategy and scaling plans. It's important for us to demonstrate a path to sustainable growth and return on investment for any stakeholders.

Current Stage (2025 - Pilot Phase):
We are currently in a **pilot-first deployment phase**, focusing on a handful of early customers. The objective this year is to validate the product in real environments and build reference cases, rather than to maximize revenue. We have budgeted for a net operational burn during this R&D intensive phase:

- **Revenue 2025:** Minimal – possibly from 1-3 pilot contracts, mostly likely either free or nominally paid. We might see low 6-figure revenue by end of year if a pilot converts to a paid small deployment in late 2025.
- **Expenses 2025:** Mainly development and operational. We have ongoing R&D (salaries for developers, AI engineers like Full Stack and AI employees noted in financials, totaling approx $171k/year combined). We have infrastructure costs (cloud servers, etc., maybe $10k/month). Some one-time costs: hardware ($300k as indicated) if we set up our own datacenter resources for testing or for a specific on-prem environment. Marketing and general expenses are modest (we're not in full marketing blitz yet). The estimated burn rate is around $50-60k per month by Q4 2025, given team expansion (some hires like Berk and Çağan at $4k/mo each as per plan). This results in a **net loss for 2025** – which is expected and planned as we invest in building the product.

Figure: Financial Timeline – Projected monthly operating cost (in USD) showing a spike at project start (hardware purchase in Sep 2025) and then stable operational costs around $25k per month thereafter.

As seen above, our cost structure ramped up with an initial capital expense (hardware or setup in 2025 Q3) and then stabilizes in a plateau – representing primarily salaries and cloud costs. We are managing our burn carefully to ensure runway into the GA launch.

6–10 Month Plan to GA (2026 H1):
By mid-2026, we plan to move to **General Availability**, which means scaling sales and marketing. Financially:

- **Revenue 2026:** We aim to convert pilots into paying customers (some upgrading to Pro/Enterprise tiers), and add new customers through direct sales efforts. Possibly target 5-10 paying enterprise customers by end of 2026, with an ARR (Annual Recurring Revenue) maybe around $1M by that point. For example, if we get 2 enterprise deals at $200k each, and a few Pro tier at ~$50k each annually, plus usage, that could sum near $1M.
- **Expenses 2026:** Expenses will rise due to go-to-market spending. We'll likely hire in Sales and Marketing. Also cloud hosting costs will rise with customer usage (though these are partially offset by usage-based billing). R&D continues but at a relatively steady state. For instance, we might grow team slightly (add 2-3 engineers, maybe a sales engineer, etc.). We might plan a marketing budget for events, conferences (~$50-100k). So monthly burn might increase early 2026 then as revenue kicks, net burn narrows. Possibly still a net loss in 2026, but we anticipate by late 2026 or early 2027 to hit breakeven if growth goes as planned.

2027-2028 (Growth and Scale):
This period we project **rapid revenue growth** – likely tripling or more year-over-year as we scale sales. Market acceptance of our type of solution by now is proven, and we're capitalizing on it:

- **Revenue:** By 2027, we target reaching maybe $5M+ ARR, and by 2028 aiming for $15M+ ARR. This is aggressive but in line with capturing key accounts and expanding within existing ones (land and expand strategy). At this stage, revenue comes from a mix of software subscription and usage fees. We foresee many Enterprise tier clients by 2028 contributing large chunks of revenue.
- **Expenses:** We will invest in scaling: international expansion (if applicable), more robust customer success and support teams to handle enterprise clients, and continuous R&D (maybe developing next-gen features, which might mean keeping a strong AI research team). However, by late 2027, we expect

operational leverage to kick in – meaning revenue growth outpaces expense growth. Cloud costs scale with usage, but we'll optimize and perhaps shift some clients to on-prem which reduces our hosting cost. Sales and marketing as a percentage of revenue will start to drop as word of mouth and market presence grows.

We will watch key **financial metrics**:

- *Gross Margin:* Should improve as we optimize infrastructure and as recurring software license revenue grows (software has high gross margin). Even though AI inference has a cost, we might incorporate that in pricing.
- *Net Margin:* We expect to move from negative in early years to positive by around 2027 or so. Our aim is to show a credible path to profitability while balancing growth; likely we reinvest heavily until we feel we have a comfortable market share.
- *CAC (Customer Acquisition Cost) vs LTV (Lifetime Value):* Early on, CAC will be higher (a lot of effort to get first few customers). But if we deliver value, retention will be high (these solutions stick once integrated). The LTV of an enterprise customer can be very large (multi-year, expanding usage). Our financial model targets a LTV:CAC ratio of >3 (meaning for every $1 spent acquiring, we get $3+ in value over time).
- *Runway and Funding:* We likely would raise funds (if a startup scenario) to cover the initial burn through 2025-2026 until we hit cashflow positive. We might plot that a Series A or B injection happens in 2025 to fund growth. The *Use of Funds* (next section) details how we allocate that.

2029 and beyond (Expansion/Maturity):
Assuming execution goes well, by 2029 Dingl AI could be a significant player in enterprise AI with healthy revenues, possibly considering exits or massive scaling (IPO, acquisition, etc.). Financially, if we continue at, say, 50-100% YoY growth through 2029, we could be looking at $30M-$50M ARR by 2029. At that scale, net profit could be achieved or we might still choose to invest in capturing more market globally. The market size being large allows for big ambitions (some players in adjacent spaces have become unicorns with 9-figure revenues). Our roadmap keeps optionality: either reach profitability and sustain or push for hypergrowth if we see the chance to become a dominant platform globally.

We included a simplified financial timeline in our planning to highlight the ramp:

- **2025:** Investment phase – spend on building, small revenue.
- **2026:** Market entry – revenue starts, still investment mode but narrowing losses.
- **2027:** Growth – revenue significant, approaching breakeven.

- **2028:** Scale – likely break even or profitable, depending on strategy; possibly reinvest in new markets.
- **2029:** Expansion – strong revenue, profits or strategic reinvestment.

Risks to the financial plan include slower adoption (sales cycle taking longer – which we mitigate via strong pilot results and customer references) and higher costs (AI infrastructure cost spikes – mitigated by optimizing model usage and leveraging cost declines as technology improves, also passing some cost to pricing). We keep an eye on gross margin to ensure our pricing covers compute costs comfortably.

In essence, our financial roadmap shows a classic SaaS trajectory: initial burn to build and acquire customers, then scaling revenue with improving margins, leading to a highly profitable software business once at scale. Given the large market and our competitive advantages, the opportunity is to capture significant value which reflects in strong top-line growth and eventual bottom-line success.

Risks & Mitigations

Every innovative venture comes with risks. Identifying these early and planning mitigations is crucial for Dingl AI's success, especially as we deploy critical enterprise solutions. Below we outline key risks across product, market, and execution, along with our strategies to mitigate them:

1. Technical Risks:

- *Model Accuracy and Reliability:* There is a risk that the AI (OCR or LLM) might not be accurate enough in certain cases – e.g., misreading documents or giving incorrect answers (hallucinations). This could erode user trust or cause errors in business processes. **Mitigation:** We use a human-in-the-loop approach for critical tasks – e.g., a validation step where a human can review AI-extracted data before it's final for high-stakes documents. We also continually fine-tune and test the models on customer-specific data to improve accuracy. We will set up a feedback mechanism where if the AI is unsure or confidence is low, it flags for manual review rather than guessing. Additionally, we'll do rigorous QA in pilot stage: measure accuracy, and not deploy fully autonomous in areas until a threshold is met. Use of proven OCR engines and state-of-the-art models plus our combination of vector+graph (which cross-checks consistency) all help ensure reliability.
- *Integration Complexity:* Dingl AI touches various systems (databases, possibly existing ERP, etc.). Integration can pose risks – data may be siloed or require custom connectors which could delay deployment or cause issues. **Mitigation:**

We designed for modular integration (clear APIs, use of standard formats like JSON, CSV). We will provide pre-built connectors for common systems (like a connector for SAP documents, SalesForce attachments, etc.). We also plan professional services or partner with system integrators to handle custom cases. Starting with clearly defined scope in pilots (not trying to integrate with everything at once) will reduce complexity, then expand integration gradually.

- *Scalability and Performance:* As data volume grows, our system might face performance bottlenecks (e.g., slow query responses if millions of documents, or heavy load on the LLM causing latency). **Mitigation:** Our architecture was chosen for scalability – Qdrant and Neo4j are scalable tech. We will do load/stress testing and ensure we can scale horizontally (e.g., sharding the vector index, clustering the DB). For the LLM, we may incorporate caching of frequent query results, or model distillation (smaller models for simpler queries). Also, we can scale by distributing workload (for instance, break huge query tasks into subtasks). On the engineering side, we will monitor system health and employ auto-scaling in cloud deployment.
- *Data Security Risk:* Handling sensitive corporate data means any vulnerability or breach would be catastrophic for trust and legal compliance. **Mitigation:** We prioritize security in development (secure coding practices, penetration testing, compliance with security standards). All data is encrypted at rest and in transit. For SaaS, we isolate each tenant's data. We also offer on-prem to those who prefer their own security. Internally, we implement strict access controls (even our team doesn't access client data unless required for support and with permission). Regular security audits and possibly obtaining certifications (SOC2, ISO27001) are part of roadmap. This assures clients and reduces risk of incidents.
- *Dependence on Third-Party Components:* We rely on open-source tools (like Qdrant, LangGraph). If any have bugs or if an update breaks something, that's a risk. **Mitigation:** We manage versions carefully (don't upgrade in production without testing). We also contribute to those communities to fix issues. Our architecture is somewhat interchangeable – e.g., if Qdrant had a big issue, we could switch to an alternative vector DB (Pinecone, etc.) since we have an abstraction layer. Diversifying knowledge among team and not being locked to one library reduces this risk. We maintain support contracts or relationships with key tech providers if available (e.g., Neo4j support subscription for quick issue resolution).

2. Market & Commercial Risks:

- *Slow Adoption / Long Sales Cycle:* Enterprises often move slowly; budgets might delay, or they opt to "wait and see". We risk slower revenue ramp if we can't

close deals timely. **Mitigation:** We employ a land-and-expand strategy: get a foothold via a small paid pilot or department use (lower barrier, quicker sale) and then grow usage. We also highlight quick ROI to justify budget. Using champions and references is key – we'll leverage early success stories to convince others. Also, targeting the right buyer personas (operations execs who have pain and budget) helps. We might mitigate by offering flexible pricing (if budget this year is small, start small, with contractual option to expand later).

- *Competition & Copycats:* The AI space is hot; big players could quickly roll out similar features, or new startups could undercut us. A large company (e.g., Microsoft or Adobe) might integrate similar capabilities into their products "for free" which is a threat. **Mitigation:** We maintain a strong **competitive moat** via our integration of features and agility. We can partner rather than fight with big players (e.g., integrate with Microsoft's ecosystem rather than always compete – maybe an add-on to Office). We focus on certain vertical expertise where we build domain-specific features that generic players won't have (like specific compliance rule sets for supply chain). Also, provide superior customer experience and customization – large generic solutions might not offer the level of tailoring we do. Another mitigation is building an IP portfolio (patents, etc.) around our unique tech, though in software patents can be tricky – still, trade secrets and head start are advantages.
- *Economic Downturn:* If the economy dips, companies cut budgets, starting often with experimental projects. Dingl AI could be seen as innovative but not essential, risking cancellations or slower sales. **Mitigation:** Emphasize cost-saving aspects of our product. If we clearly show we save money (automation replacing repetitive tasks), then even in downturn, we help them cut costs. So marketing messages might shift from "innovate with AI" to "save $$ with AI by reducing manual workload" in tougher times. Also, offer flexible models – if they can't do capex, do opex via subscription; if short on cash, maybe usage-based that scales down. We ensure our own cash management is prudent to weather slower periods. Having some customers in recession-proof sectors (like certain government or resilient industries) can balance risk too.
- *Client Change Management Risk:* Sometimes the risk isn't the tech, it's getting the organization to adopt it. Employees might resist (fear of job loss due to automation) or not use the system fully, leading the project to fail to deliver value. **Mitigation:** We include strong customer success programs. That means training end-users, providing easy-to-use interfaces, and perhaps championing a "augmentation not replacement" narrative to user base. We help management communicate the benefits. Starting with co-pilots rather than fully autonomous processes can ease people in. We also design the UI to be user-friendly and integrated into their daily tools (maybe integrate with email or Slack for alerts,

etc., so they don't have to go to a new place). Basically, drive adoption by minimizing behavior change needed and highlighting user benefits (like "this makes your job easier, not redundant").

3. Execution Risks (Internal):

- *Scaling the Team and Operations:* As we grow, hiring the right talent (AI engineers, sales, support) is critical. There's risk we might not hire fast enough or on-board them well, leading to execution issues. **Mitigation:** We create a strong culture and employer brand (exciting problem, modern tech) to attract talent. We might use strategic hiring (like ex-colleagues, advisors networks). Also, we can outsource or partner for some aspects if needed (e.g., use a cloud service for some part rather than build in-house if short on dev capacity, or channel partners to aid in sales). Clear documentation and processes internally mitigate losing key person risk.
- *Regulatory Changes:* AI is coming under regulatory view (data privacy laws, AI bias and transparency regulations). If new laws require certain handling (e.g., data can't be used to train without consent, or certain AI decisions need explainability), we must adapt or risk non-compliance. **Mitigation:** We keep a legal/regulatory watch (maybe have an advisor for this). We design with privacy in mind from the start (no personal data leaks, etc.). Our knowledge graph can actually help with explainability (we can trace how an answer was formed from sources – a plus if regulators ask). We're ready to implement features like audit logs of AI outputs, option to disable learning on sensitive data, etc. Being proactive here turns a risk into a potential selling point (a compliant AI platform).
- *Dependency on Founders/Key Individuals:* Early on, a lot of knowledge and decision-making may reside with founders or a few key engineers. That's a risk if someone is out or leaves. **Mitigation:** Spread knowledge via code reviews, documentation, and a strong team culture of collaboration. Also, perhaps offer incentives to key members to stay (stock options, etc.). Ensure multiple people can handle each major area.

4. Client & Project Risks:

- *Pilot Fails to Deliver ROI:* If our early deployment doesn't show clear benefits (maybe due to unforeseen complexity or insufficient data quality on client side), we risk losing that client and negative word-of-mouth. **Mitigation:** We select pilot projects carefully (with high chance of success, supportive client team). We set realistic goals and KPIs up front and monitor them. If something isn't working, we pivot quickly or add manual support to bridge gaps (even if we do some manual backend work in pilot to ensure outcome). Essentially, we do what it takes to make first customers happy, using a "white-glove" approach.

- *Liability Risk:* If Dingl AI makes an error that leads to a serious business mistake (e.g., it misses a compliance issue and client gets fined), there's risk of liability or at least loss of customer. **Mitigation:** Our contracts will limit liability (standard in enterprise software). We also strongly position the product as decision-support, with humans still accountable, especially in early phases. Over time as trust builds, they'll lean more on it, but by then accuracy will be proven. We also encourage phased rollouts (parallel run with old process to compare results before fully switching to us). Insurance (errors & omissions insurance for AI advice) could be considered as a backstop.

In conclusion, while the risks are diverse – technical, market, operational – we have a **mitigation plan for each**. By being proactive and customer-focused, we turn many of these risks into areas of strength:

- Emphasizing accuracy and transparency addresses AI reliability concerns.
- Flexibility in deployment and pricing addresses adoption hurdles and competition.
- A continuous improvement mindset (in models and in customer support) ensures we adapt quickly to any issues.

We will maintain a risk register and review it periodically as we grow, ensuring new risks are caught and strategies updated. Our investors and customers can take confidence that we're not blind to these challenges; rather, we have concrete plans to navigate them, giving Dingl AI the best chance to thrive in a dynamic environment.

KPI Scorecards

To ensure Dingl AI is delivering value and to guide our internal performance, we will track a set of **Key Performance Indicators (KPIs)**. These KPIs span both business metrics and product usage metrics. We present them in scorecard form to monitor progress over time and to show stakeholders the impact of our solution.

Operational/Business KPIs:

- **Customer Acquisition & Retention:**
    - *Number of Pilots in Progress:* (Target: e.g., 5 pilots by end of Q2 2026) – Indicates market traction. We expect this to climb during initial go-to-market then convert to paying customers.
    - *Conversion Rate from Pilot to Paid:* (Target: 80%+) – If pilots are successful, a high rate means our sales funnel is effective and product is proving value.

    - *Customer Retention Rate (Logo Retention and Revenue Retention):* We aim for >90% logo retention annually (i.e., very few customers drop off) and >100% revenue retention (meaning upsells lead to more revenue from existing base even before new sales). This will be crucial in showing that once adopted, customers stick and expand usage.
    - *Net Promoter Score (NPS):* We'll periodically survey users. Target NPS might be 50+ which is excellent for B2B software – indicating users would recommend us. A high NPS would reflect user satisfaction and gives qualitative feedback too.
- **Financial KPIs:**
    - *Monthly Recurring Revenue (MRR) / Annual Recurring Revenue (ARR):* The growth of ARR is the prime business metric. We might have milestones like ARR of $1M by end of 2026, $5M by 2027, etc. The KPI scoreboard will track actual vs target.
    - *Gross Margin:* Target 70-80% in the long run. Early on it might be lower due to heavy compute costs relative to low user count, but as we scale and optimize, we expect improvement. Gross Margin reflects profitability of delivering service.
    - *Burn Rate and Runway:* Internally, we'll watch monthly burn and how many months of cash runway remain (especially if venture-funded). We aim to manage burn such that we always have ~18+ months runway or are nearing profitability by the time we'd drop below that.
    - *Cost per Document Processed:* An efficiency metric – our internal cost (cloud compute etc.) for each document page processed. We want this to decrease over time via optimizations. If initially it's, say, $0.02, we try to bring it down (which improves margin or lets us cut price if needed).

Product Usage & Performance KPIs:

- **Processing Efficiency:**
    - *Documents Processed per Day:* This indicates scaling usage. We might plot daily/weekly throughput. It should climb as we onboard more clients and as clients expand usage. Also, if a single client ramps up, it tests system scalability. We'll set a system-wide target (like capable of processing 100k pages/day by such date).
    - *Average Processing Time per Document:* From ingestion to result ready. If currently a document takes 5 minutes to go through pipeline, we aim to reduce that to say 2 minutes on average through optimizations and better parallelism.
    - *AI Query Response Time:* We'll track how long it takes to answer user queries (the LLM/agent). For good UX, target could be <5 seconds for

typical queries. We'll watch 90th percentile latency and keep it within acceptable range by scaling infra.

    - *Accuracy Metrics:* We'll maintain a "quality scorecard" from validation data: e.g., OCR accuracy (% characters correctly recognized), field extraction accuracy (if we predicted fields, how many correct), and Q&A accuracy (perhaps measured by user feedback or a test set of questions). Targets maybe like >95% on critical field extraction. Over time these should improve with model tuning. This is crucial to demonstrate improvement and reliability.
    - *Automation Rate:* For processes where Dingl AI is deployed, what percentage of documents or tasks are fully automated vs requiring human intervention. E.g., if in invoice processing 70% go through without human touch, that's the automation rate. We want this rate to climb as system gets smarter. It's a direct measure of value delivered (higher automation = more savings).
- **User Engagement:**
    - *Active Users:* How many distinct users use the system per week/month. High active users means good adoption. We might measure both end-users (analysts querying) and admins (setting pipelines). We want to see growth in both as we expand in accounts.
    - *Queries per User:* On average, how many questions are users asking the AI assistant. If this increases, it implies they trust and rely on it more. We might see, say, an analyst going from using it once a day to perhaps 10 times a day as it becomes integral to their workflow.
    - *Feature Usage:* Are they using key features? e.g., the % of customers using custom pipelines (Pro feature uptake), or the usage of the fine-tune feature (which indicates advanced usage). If certain features are under-used, we investigate why (maybe need better training or UX improvement).
    - *Support Tickets/Response Time:* Indirectly, how often users need to contact support, and our responsiveness. Fewer tickets per customer indicates product is intuitive/stable; fast response indicates we handle issues well. We target say first response within 1 hour for high priority in Enterprise, etc., and track adherence.

We will compile these KPIs into **scorecards** perhaps segmented by client and overall:

- For each client, a dashboard could show "documents processed, time saved (estimated), issues flagged, etc." – great for showing ROI to them.
- Internally, we have our growth and performance scorecards updated monthly.

For our investors/board, we might highlight:

- Sales & marketing efficiency (LTV/CAC as earlier, pipeline deals, etc.).
- Development velocity (e.g., number of releases or new features delivered per quarter – a KPI for our R&D efficiency).

One specific KPI that's powerful in our context is **"Time Saved for Clients."** We can estimate that by measuring how long these processes took before vs now. For example, if 10,000 documents were processed by Dingl AI in a month and each would've taken 5 minutes of human time, that's ~833 hours saved. We can aggregate and present that as a key value delivered (maybe even put it on our website). It's not a traditional SaaS metric, but it resonates with business value.

Another is **error reduction** – if we have data, we could track error rate in processed docs (like discrepancies found). If originally 5% of documents had human errors and now it's 1%, we quantify that improvement.

We will likely maintain an internal Balanced Scorecard covering:

- Financial (ARR, burn, etc.),
- Customer (NPS, retention, usage growth),
- Internal processes (deployment times, model accuracy),
- Innovation/Learning (employee metrics, R&D milestones).

By consistently tracking these KPIs, we ensure we remain outcome-focused. They will alert us early to any issues (e.g., if active usage isn't growing, why? If accuracy dips after a model change, we catch it). They also provide great material for marketing (high NPS, big time savings) and for continuous improvement as we scale Dingl AI.

Use of Funds

For a venture like Dingl AI, effective use of capital is critical to achieve our milestones. Based on our financial plan and assuming we raise (or allocate) funding for the next phase of growth, here is how we intend to use the funds:

1. **Product Development (R&D) – ~40% of funds:** This is our top priority investment. It includes salaries for our technical team (AI engineers, full-stack developers, QA). Key uses:
    a. Continue improving the core platform: refining the pipeline, integrating additional data sources.
    b. Advancing AI models: fine-tuning domain-specific models, developing improved OCR or adding support for more languages, etc.

c. Scalable architecture: spending on devops tooling, cloud infrastructure setup to ensure reliability (some of this overlaps with infra costs).
d. New features: e.g., building out more advanced analytics dashboard, user management features, etc., based on pilot feedback.
e. IP creation: any funds towards research or patents to protect our innovations falls here too.

   We've allocated a significant portion here because a strong product will drive market success. For example, if we raised $2M, around $800k might go to R&D efforts over the next 18 months, which aligns with expanding the dev team and covering tech expenses.

2. **Infrastructure & Operational Costs – ~15%:** This includes the cloud hosting costs for our SaaS environment, purchasing any necessary hardware (as we did initially for development or for on-prem deployments support). For instance, we spent $300k on hardware as optional in our plan (maybe for an on-prem lab or initial data center, as shown in finances). Going forward, we'll likely need to budget for:
   a. Cloud hosting for pilots and early customers (which scales with usage).
   b. Dev/Test environments, possibly some specialized hardware (GPUs) for model training and fine-tuning.
   c. Operational software (monitoring tools, security tools).

      While we'll try to cover hosting with revenues (usage fees), early on we may subsidize some pilot hosting, so having funds earmarked avoids needing to charge prematurely or skimp on performance.
3. **Sales and Marketing – ~25%:** Once the product is ready for GA, we need to fuel our go-to-market. Funds here will:
   a. **Sales Team:** Hire sales executives or solution consultants, especially who have domain knowledge in our target industries. This also covers sales commissions, travel for enterprise sales (demos, on-site meetings).
   b. **Marketing Campaigns:** Content marketing (whitepapers, case studies), attending or sponsoring industry conferences, online ads targeting decision makers, and running webinars or workshops. We may allocate funds for pilot programs or POCs (e.g., offering a subsidized pilot to key logos as marketing expense).
   c. **Partnerships:** Possibly funds to integrate with or partner with bigger players (for example, paying to be a certified partner in a tech ecosystem, or small integrations).
   d. **Brand development:** Designing a professional website, brand materials, etc., maybe engaging a PR firm for media placements once we have success stories.

      For example, out of a given budget, we might spend $50k on a major trade show, $30k on digital lead generation, hire 2 salespeople at $120k each,

etc. This category is crucial to start filling our sales pipeline and accelerating adoption beyond referrals.

4. **Customer Success & Support – ~10%:** Ensuring pilot customers and early adopters succeed is crucial for retention and references. Funds here go to:
    a. Hiring customer success managers or solution architects who assist clients in onboarding, training, and customizing pipelines.
    b. Setting up support infrastructure (ticketing system, knowledge base content creation).
    c. Perhaps an online community forum or user conference down the line. Happy customers will become case studies that feed back into marketing, so it's an important investment. This might involve one or two key hires in this area initially and scaling as customers grow.
5. **General & Administrative – ~10%:** This covers legal, finance, admin, and overhead:
    a. Legal fees for contracts, possibly patents, compliance (especially if we need specific certifications).
    b. Accounting and bookkeeping, maybe hiring a finance person or using external CPA.
    c. Office expenses (though many startups are remote or modest here), or software tools for internal productivity.
    d. If applicable, loan repayments or other prior obligations (though ideally minimal).
    We keep this lean but it's necessary to keep operations smooth. For instance, we might need to spend on insurance (E&O insurance given we deliver enterprise software, liability coverage, etc.).

【Use of Funds Chart Placeholder】 *A chart could illustrate these percentages visually – e.g., an allocation pie: 40% R&D, 25% Sales/Marketing, etc., to communicate spending priorities.*

The above breakdown is an initial guideline. As we progress, we'll reallocate as needed. For instance, if product development is ahead of schedule but sales pipeline is slow, maybe shift more to marketing to boost leads. Or if we find infra costs are rising faster with user growth, ensure pricing and cost control to adjust.

Timeline of Fund Usage:

- In the first 6 months post-funding, heavier on product development (finish features, pilots), moderate marketing (teaser campaigns, getting initial leads).
- Months 6-12: ramp up marketing and sales spend as product hits GA, while still continuing R&D (maybe slightly less % as marketing increases).

- We anticipate possibly another fundraise after proving product-market fit to really scale sales globally (that would fund a larger salesforce, international expansion, etc.). The current funds aim to get us through product launch and initial scale.

Return on Investment: Using funds in the above manner should allow us to:

- Build a robust product that wins pilot trials (R&D spend yields a competitive offering).
- Acquire our first paying customers and generate revenue (sales/marketing spend yields ARR, at an acceptable CAC).
- Keep customers happy (customer success spend yields high retention and upsells).
- All while managing the backend operations securely (ops/admin spend yields stability and compliance, avoiding costly incidents).

We will keep a close eye on metrics like CAC payback period (how many months to recoup customer acquisition cost via subscription revenue – we aim for <12 months ideally), and operational burn relative to revenue (to gauge when we hit profitability or need next funding).

Finally, we maintain some buffer for unforeseen needs – perhaps ~5-10% of funds unallocated as contingency. Startups often face surprises (maybe a sudden need to pivot a feature, or an opportunity to do a strategic project) – a buffer ensures we can respond without jeopardizing core plans.

In summary, our planned use of funds is balanced to drive both product excellence and market penetration, while supporting the business infrastructure needed. By investing significantly in R&D and customer-facing functions (nearly 2/3 of funds combined), we focus on creating and delivering value – which in turn should translate to revenue growth and a path to profitability, providing returns to investors and sustaining the company's mission to transform enterprise data workflows.