Project Plan: LLM-Based Compliance Checking Tool for Data Entry

1. Introduction and Overview

This document outlines the plan for developing and integrating an LLM-based tool into Vistavu's existing software system. The primary objective is to enhance compliance checking during data entry by leveraging Large Language Models to extract, process, and apply rules derived from contracts and legal documents. This will provide real-time guidance to users, ensuring adherence to regulations regarding contract worker wages and service-related expenses.

2. Problem Definition

The current data entry process requires manual interpretation of complex contracts and legal documents to ensure compliance with wage and expense regulations. This process is time-consuming, prone to human error, and can lead to inconsistencies or violations. The challenge is to automate the extraction of these rules and integrate them seamlessly into the data entry workflow to proactively guide users and minimize non-compliance.

VistaVu is partnering with Edmonton Exchanger (EdEx) to address these core issues. Their current system, which includes manual data entry on crew sheets, leads to significant challenges in billing and reconciliation due to discrepancies between EdEx's internal records and customer systems. The proposed tool will solve this by automating the rule extraction process, using a standardized "rule card" format, and applying these rules in real-time to data as it is entered. This will minimize manual effort, reduce financial risk, and improve overall reporting accuracy.

3. Dataset

The development of the LLM-based tool will rely on a comprehensive dataset comprising:

- **Contracts Documents:** A diverse collection of the client's existing contracts, such as the National Maintenance Agreement, union and non-union agreements, customer agreements (like Shell, Imperial Oil), and regulatory documents that govern contract worker wages and service expenses.
- **Historical Top Problems:** Samples of past data entries, including crew sheet samples and instances of non-compliance, to understand common errors and validate the effectiveness of the tool.
- **Annotated Rules:** A subset of the contracts documents with manually identified and extracted compliance rules to serve as ground truth for model training and evaluation.

4. Proposed Method

Our proposed method involves a multi-step approach for developing the LLM-based compliance checking tool. It includes ML Workflow, Development and Implementation Process, and a Workflow tool at deployment.

4.1. Approaches of building an LLM Application

This section outlines the complete machine learning lifecycle that the client will experience, from the moment a new document is introduced to the final real-time validation within the data entry system. This is a continuous process designed to ensure that the rule base is always up-to-date and that users are consistently guided by the correct and most current regulations.

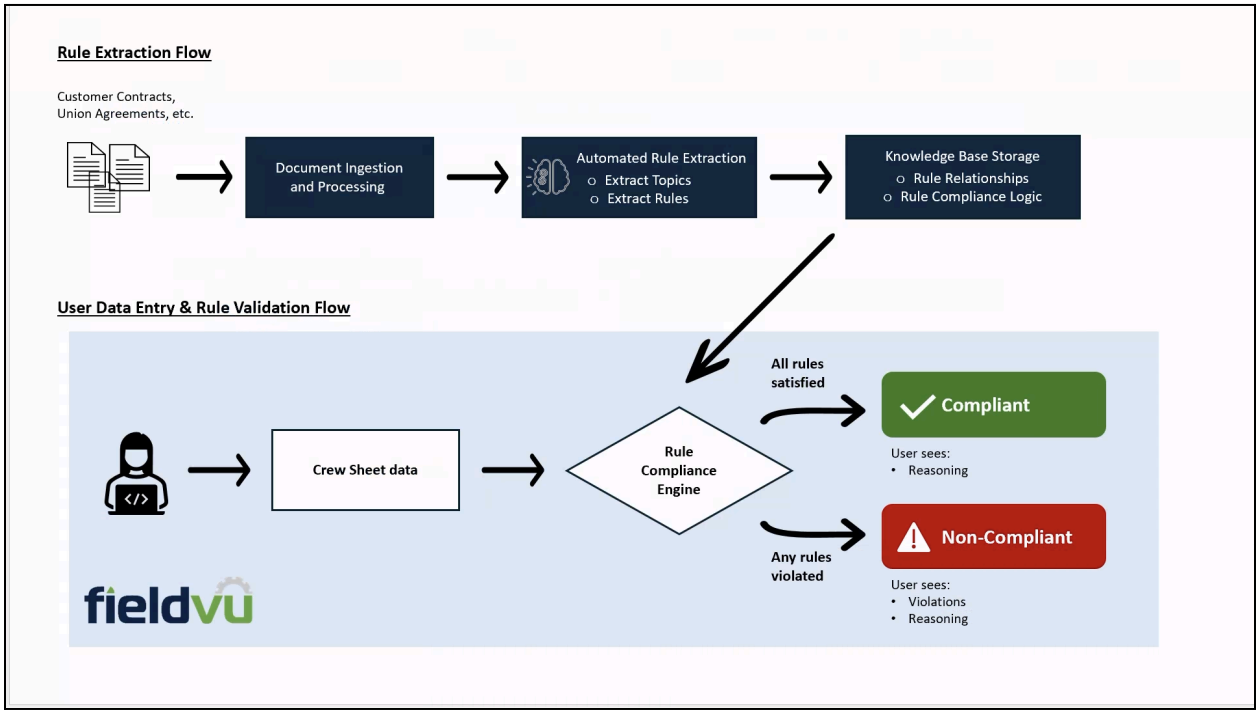

Figure 1: Workflow representation of compliance checking

1. Document Ingestion and Processing: This initial phase is about transforming raw, unstructured documents into a format that the LLM can understand and process effectively.

- **Document Conversion**: The first step involves converting various document types (e.g., PDFs) into a uniform, structured format like plain text. This is crucial because LLMs operate on text data.
- **Classify Labour-Related Context : T**he goal here is to drastically reduce the 43 pages of the NMA document to just the Articles that contain rules related to labour for validation and then chunk them logically.
- **Intelligent Chunking**: This is a sophisticated method of splitting the document's content. Instead of a simple split by a fixed number of words or characters, intelligent chunking analyzes the document's structure to create semantically meaningful sections. For

contract documents, this means identifying logical break points like headings, subheadings, or the end of a clause.

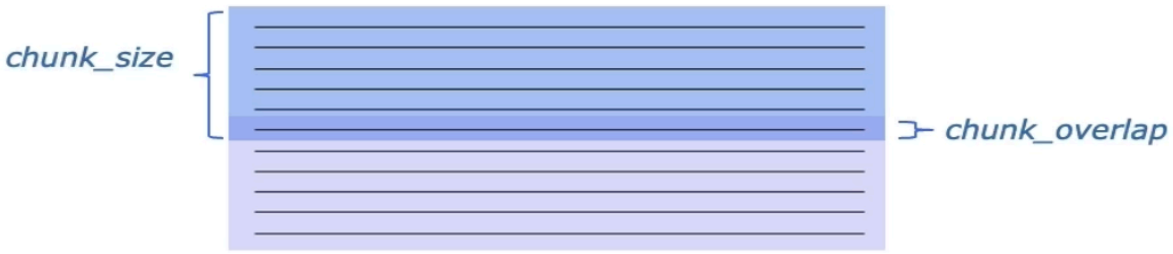

Figure 2: Splitting Text with Chunk_overlap (Taken from LangChain documentation)

Phase 2: Knowledge Base Storage (Topic and Elemental Rule Extraction)

This phase establishes the foundational structure for the rule validation engine by defining a shared rule schema across all three documents (NMA, Boilermakers, NWR Contract).

Step 2.1: Topic Extraction and Topic Rule Card Creation

- **Process: The LLM identifies and establishes a single, unified set of Topic Rule Cards (e.g., `TOPIC_HOURS_001: Labour Hours`, `TOPIC_MEAL_002: Meal Provisions`) based on semantic analysis of all retained content.**
- **Rationale: This fixed, high-level granularity allows the validation engine to efficiently use timesheet context (e.g., the worker's trade or location) to instantly select the most relevant topics for checking.**

Step 2.2: Elemental Rule Card Creation

- **Process: The LLM aggregates content from all relevant articles for a single Topic (the Topic Summary) and formally defines the granular, single-logic Elemental Rule Cards (Nodes) that apply across all three documents.**

Step 2.3: Code Conversion Point 1: Elemental Rule to Formal Logic

- **Process: The LLM converts the natural language `rule_excerpt` (e.g., "entitled to a meal break after ten hours") into a structured, executable Formal Logic expression.**
    - **Input Example: "When an employee works beyond ten (10) hours per day..."**

    - Output Logic: `if context.hours_worked > 10 and context.schedule_type == 'daily':`
- **Rationale: This is the first critical translation from human language to machine-testable logic, defining the unambiguous Criteria (Implied) field for the rule card.**

Step 2.4: Variable Definition

- **Process: Define and standardize all necessary input variables required to test the Formal Logic across all three documents (e.g., `context.work_schedule`, `context.is_camp`, `context.is_callout`).**
- **Rationale: Standardizing variables ensures interoperability and consistent data input when the final validation function is executed.**

Step 2.5: SME Vetting (Human-in-the-Loop)

- **Process: Subject Matter Experts (SMEs) rigorously review the extracted Elemental Rules and validate the Formal Logic (Code Conversion Point 1) produced by the LLM for all three documents simultaneously.**
- **Rationale: Ensures the extracted logic accurately reflects the legal intent and correct granularity before proceeding to the complex dependency modeling.**

Phase 3: Directed Graph Construction and Logic Compilation

This phase models the complex interactions between rules and compiles the final executable program.

Step 3.1: Graph Relationship Building

- **Process: Formally link all Elemental Rule Cards (Nodes), defining internal dependencies within each topic (e.g., `R2.Meal_Break` `FOLLOWS` → `R1.Hours_Check`) using Directed Graph edges.**
- **Rationale: This creates the base decision flow for each topic, modeling the sequence in which logic must be checked.**

Step 3.2: Cross-Document Linking (Conflict Resolution)

- **Process: The LLM identifies and defines external relations between rules found in different documents.**
    - **Example Link: NWR Customer Contract rule → `OVERRIDES` → Boilermakers Union rule.**

- **Rationale: This resolves complex conflicts and ensures the resulting system always applies the highest-priority, legally binding rule based on the specific job context.**

Step 3.3: Missed Link Scan

- **Process: The LLM performs a targeted semantic search across the full, filtered text of all three documents to capture any contextual overrides, exceptions, or special rules (e.g., an obscure clause defining callout).**
- **Rationale: Acts as a final quality control layer to ensure no edge cases or subtle dependencies are missed before compilation.**

Step 3.4: Code Conversion Point 2: AST Creation and Auto-Generation

- **Process: The Graph-to-AST Compiler takes the fully enriched Directed Graph and automatically generates the complete Abstract Syntax Tree (AST). This AST is then used to auto-generate the project-specific, highly optimized, executable timesheet validation code (a single, small Python function).**
    - **Input: The Directed Graph (the network of all linked rules).**
    - **Output: Executable function code.**
- **Rationale: Compiling the graph guarantees that the final code executes all logic in the guaranteed correct order, providing speed, precision, and a fully auditable path.**

Step 3.5: Real-Data Testing

- **Process: The generated validation function is rigorously tested against real-world and synthetic timesheet entry examples, covering a balanced set of correct, violating, and edge-case scenarios (as defined in the Edex data requirements).**
- **Rationale: Final functional verification to confirm that the compiled AST accurately reflects all inter-document rules and correctly determines compliance status and violation reasons.**

4. Real time guidance: When a user begins a new data entry, the system instantly analyzes the entered information, such as the customer, jurisdiction, and charge category. It uses these details to intelligently filter the thousands of rules in the knowledge base, retrieving only those that are directly applicable to that specific entry. This ensures that the user receives guidance based on the most relevant and current regulations.

5. Real-time Compliance Validation: This is where the user directly interacts with the system. As data is entered, the tool works behind the scenes to **intercept the information**. It then dynamically queries the knowledge base to retrieve the specific rules applicable to that entry. Validation is performed by a **Python code interpreter** that executes code generated from the

compliance criteria. This process ensures a reliable and objective check of the data against the predefined rules, providing immediate feedback to the user.

6. Immediate User Feedback: The final step is to communicate the results to the user in an easy-to-understand way.

- **Contextual Warnings**: If the code interpreter returns `false` (indicating non-compliance), the system triggers a pop-up or a highlighted field on the user interface.
- **Detailed Explanation**: The feedback isn't just a simple "Error." It pulls the `Source` and `Summary` from the applied Rule Card to explain **why** the data is non-compliant. For example, it might say: "Warning: Your entry for Overtime Hours exceeds the maximum allowed by the **NMA agreement (Section 3.1.2)**, which states that all hours over 8 per day are considered overtime." This level of transparency builds user trust and helps them learn the rules.

Prompt Engineering & Plugin Strategy

We will fine-tune the LLM's performance using **few-shot learning**, providing it with a small, curated set of examples to guide its extraction process. A key strategic decision is to use a **plugin method** for the rule extraction criteria. By treating the criteria for extracting labor-related rules as a variable, we can easily swap it out for new criteria in the future (e.g., criteria for materials or equipment rules), enabling seamless upscaling without a complete redesign.

4.3. Workflow of the Tool at Deployment with an Example

This section describes the practical, step-by-step process of how the LLM-based compliance tool operates in a live environment, from the moment a user begins a data entry task to the final storage of the compliant record. This workflow is the initial proposed idea and will be further refined as per requirements.

The Story: Data Entry as the Validation Trigger

The workflow begins when a timekeeper is preparing the Crew Sheet for the day. For our example, the timekeeper must manually enter all the crucial operational context: the worker is a **Pipefitter-Journeyman**, the job is under the **NMA-AB** agreement, they worked a 15-hour shift on a Wednesday (a **5x8 schedule**), and the site involved **Camp accommodations**. As the timekeeper manually fills in these fields, the system collects this entered data, which acts as the **Context Attributes** necessary to select and execute the correct compliance logic. The goal is to instantaneously validate the submitted pay breakdown (ST/OT/DT) and entitlements (meal allowance) against the complex rules derived from the multiple labor documents.

Step 1: Context Capture and AST Loading (The Trigger)

The moment the timekeeper enters the mandatory contextual data—such as **Employee Trade, Total Hours, Schedule Type, and Accommodation Status**—the system captures these inputs. This complete set of entered data (e.g., `trade: Pipefitter`, `hours: 15`, `is_camp: TRUE`, `document: NMA-AB`) is immediately used to perform a rapid **Topic Filter**. Instead of checking thousands of potential rules, the system only loads the necessary pre-compiled code. Since the entry involves a Pipefitter and a 15-hour shift, the system loads the **Overtime Hours AST** and the **Overtime Meals AST**, significantly reducing validation time.

Step 2: Execution of the Overtime Hours AST

The system first executes the simplest and most foundational AST—the **Overtime Hours AST**. This AST, which was generated from the **Formal Logic** of the NMA-AB's standard hours rules, runs instantly. The compiled logic follows its guaranteed execution order to determine the required pay breakdown for a 15-hour shift on a 5x8 schedule: eight hours of Straight Time (ST), followed by two hours of Time and One Half (OT), and finally, five hours of Double Time (DT). The logic establishes the definitive **Required Hours**: **8 ST, 2 OT, 5 DT**.

Step 3: Execution of the Overtime Meals AST (Traversing the Graph)

Next, the system executes the more complex **Overtime Meals AST**. This code block traverses the logic flow dictated by the **Directed Graph** of linked Elemental Rule Cards:

1. **Primary Obligation Check (R2.Meals):** The compiled code immediately executes the primary rule (R2.Meals, Art. 15.600) based on the input that **15 hours were worked**. This establishes the initial liability—the **primary obligation** to provide a company meal and a paid break.
2. **Substitution Check (R3.Allowance):** The logic then checks the substitution rule (R3.Allowance, Art. 15.602). In case a meal was not provided, this rule substitutes the meal obligation with a **$40 meal allowance** and a longer paid break.
3. **Override Check (R4.Camp):** This is the crucial **Cross-Document Linking** step. The code executes the Camp Override Check (R4.Camp, Art. 15.604). Because the timekeeper entered **Camp accommodations**, the logic follows the graph's **OVERRIDES** link, executing the rule that **explicitly removes the obligation to pay the $40 meal allowance**.

Step 4: Final Reconciliation and Feedback

The system compares the compiled final obligations to the **user-entered data** in the crew sheet:

- **Hours Reconciliation (FAIL):** If the timekeeper entered a breakdown of 9 ST, 3 OT, and 2.5 DT, the system compares this against the required 8 ST, 2 OT, and 5 DT. The system instantly returns a **FAIL** status.

- **Meal Reconciliation (PASS):** Since the compiled logic determined that **no meal allowance** is owed due to the **Camp Override**, if the timekeeper did not enter a meal allowance payment, the entry is flagged as **PASS** on meal entitlement.

The system returns a concise, actionable error message to the timekeeper, allowing them to correct the hours breakdown before submission. Crucially, the system preserves the full **Auditable Path**, logging the sequence of elemental rules and overrides (R2 → R3 → R4) that were followed to arrive at the final compliance decision.

5.Development Process: Three-Stage, Evaluation-Focused Approach

Our development methodology is a focused, three-stage approach designed to de-risk the project by guaranteeing the precision of rule extraction before committing to code generation. This process is fundamentally powered by a continuous **Human-in-the-Loop (HIL)** evaluation framework.

Approach 1: Establishing Ground Truth and Baseline (The Metric Foundation)

This initial stage ensures we have a reliable, quantifiable standard for success. We cannot trust the LLM's output until we can objectively measure its accuracy against validated contract rules.

- **Dataset Curation and Annotation:** We collaborate directly with your Subject Matter Experts (SMEs) to manually annotate the complex clauses in the **NMA-AB, Boilermakers, and NWR Contract**. This annotated subset of 60−70 line items (as defined in our data requirements) becomes our definitive **Ground Truth**—the only source used to verify rule extraction accuracy.
- **Research and Prototyping:** We select and fine-tune specific LLM architectures known for their strength in structured output and long-context processing. We then run initial prototypes against a non-annotated dataset subset to establish a **Baseline Performance**.
- **Evaluation Checkpoint:** The deliverable is a set of quantifiable **Baseline Metrics** (Precision, Recall, F1-score) for rule extraction against the **Ground Truth**. If the baseline is too low (e.g., F1 is below 0.80), we pause development to adjust the LLM architecture or prompt engineering until minimum standards are met.

Approach 2: Iterative Graph Generation with Human-in-the-Loop (The Core Engine Build)

This is the central development loop where the graph structure is built, rules are formalized, and the core logic is converted into structured code. This stage is governed entirely by the HIL process to capture nuanced legal intent and resolve conflicting contract rules.

- **LLM Extraction and Formal Logic Conversion (Code Conversion Point 1):** The LLM performs the extraction and conversion, moving from contract text to **Elemental Rule Cards** and then generating the **Formal Logical Expression** (e.g., Python pseudo-code) for each rule.
- **HIL Validation and Correction:** SMEs review two things simultaneously: 1) The *semantic accuracy* of the extracted rule (did the LLM miss an exemption?) and 2) The *code accuracy* of the Formal Logic (does the `if context.is_camp == true` correctly model the intent?).
    - **Feedback Loop:** Any error identified by the SME is immediately fed back as a **Correction Instance** to retrain the LLM's prompt, refining its behavior specifically for that type of rule/document context.
- **Graph Linking and Conflict Resolution:** The LLM defines all internal and **Cross-Document** links (Overrides/Substitutes). This HIL stage is critical: SMEs must explicitly confirm the precedence (e.g., "The NWR Contract rule must **Override** the Boilermakers rule for this job scope"). The verified links are then used to lock the structure of the **Directed Graph**.

Approach 3: Integration, AST Compilation, and End-to-End Validation (The Final Product)

The final phase compiles the validated logic into a high-speed validation engine and verifies its real-world function within your target environment.

- **AST Compilation (Code Conversion Point 2):** The completed, human-verified **Directed Graph** is passed to the **Graph-to-AST Compiler**, which generates the final, optimized Python function (the AST). This generation is treated as deterministic and relies entirely on the accuracy achieved in Approach 2.
- **Integration Module Development and Functionality Testing:** We build the API that takes timesheet input and calls the generated AST function. Functionality testing confirms the system's real-time performance, ensuring the validation function executes instantly and delivers accurate feedback to the FieldVu UI without latency.
- **End-to-End Validation (UAT):** We conduct comprehensive validation using the remaining live and complex data scenarios (outside the initial Ground Truth). User Acceptance Testing (UAT) focuses on **trust and usability**: Do timekeepers understand the violation messages, and do the system's compliance decisions align with the Head Office's final ruling? This final step confirms the tool is accurate, fast, and trustworthy for production deployment.

6. Potential Problems and How We Will Address Them

Potential Problem	Addressing Strategy
LLM Hallucinations & Inaccurate Rule Extraction	LLMs can generate plausible but incorrect information. We will address this with a human-in-the-loop review process for all critical rule extractions and will implement robust validation mechanisms, such as cross-referencing extracted rules with multiple sources where possible.
Difficulty with Long-Range Context & Nuance	Contract documents contain complex clauses, definitions, and cross-references that span many pages. LLMs may struggle to maintain context over extremely long documents, leading to missed connections. Our solution will use intelligent chunking to preserve semantic integrity and leverage models with large context windows to handle these dependencies.
Model Instability & Lack of Reproducibility	The non-deterministic nature of some LLMs can lead to inconsistent output, which is unacceptable for a compliance tool. We will address this by using deterministic models or specific settings and by running the same prompt multiple times to confirm the consistency of the output.
Cost & Scalability with Increasing Document Volume	Running large LLMs, especially with long context windows, can be computationally expensive. Our strategy includes optimizing LLM inference for speed and efficiency. We will also use pre-processing techniques, such as chunk classification, to minimize the amount of data fed to the core extraction model.
Data Privacy & PII Concerns	Processing client contracts and crew sheets requires strict data handling protocols. We will ensure all LLM interactions are conducted within a secure environment, with PII (Personally Identifiable Information) scrubbed from prompts before they are sent to the model. We will implement industry-standard data encryption and access controls.

User Trust & Explainability	Users may not trust an automated system that seems like a "black box." We will build confidence by providing clear, auditable explanations for every compliance check. The "rule card" design, with its detailed source and summary, will be directly referenced in the user interface to show exactly which rule was applied and where it came from.
Ongoing Model Drift & Rule Updates	Legal documents and contracts change over time, leading to rule changes that can cause LLM-extracted rules to become obsolete. We will establish a clear process for monitoring and updating the rule base and will regularly retrain the LLM models to adapt to changes in regulations and contractual language, preventing model drift.

7. Milestone

Phase	Core Goal	Time - Weeks	Key Activities (Cross-Document Focus)
I. Data Ingestion	Convert all 3 documents.	**5 Weeks**	**PDF-Text Conversion & Chunking** **LLM Classification & Filtering** **Topic Extraction:** LLM identifies a **set of Topic**
II. Elemental & Topic Rule Extraction & Alignment	Extract all granular rules and define the initial set of variables.	**9 Weeks**	**Elemental and Topic Rule Card Creation- Code Conversion Point 1:** Elemental Rule to Formal Logic **The LLM converts the `rule` into a formal, code-like expression ; Variable Definition** **SME Vetting: Human-in-the-Loop** Subject Matter Experts (SMEs) review all 3 sets of extracted Rules to ensure the boundaries and extraction fidelity are correct.
III. Directed Graph & Conflict Resolution	Define all internal and inter-document relationships to create the single network.	**4 Weeks**	**Graph Relationship Building:** Formally link all Elemental Rule Cards, defining internal relations (`FOLLOWS`, `SUBSTITUTES`). **Cross-Document Linking** **Missed Link Scan:** Perform a targeted semantic search to capture all special rules.
IV. Final Compilation & Integration	Generate the executable logic for the application.	**3 Weeks**	**AST Creation and auto-generation of project-specific rule validation code and engine.** **Real-Data Testing**
TOTAL PROJECT DURATION	**All 3 Documents Integrated**	**21 Weeks**	**~5.0 Months**

9. Appendix