From Traditional Practice to AI Architecture

— For Claims and Underwriting Professionals Ready to Lead AI System Architecture

Preface

I began my career as a resident physician after completing my clinical medicine degree, but a series of coincidences eventually led me into the insurance claims and loss adjusting profession. For more than two decades, my work has focused on applying medical expertise to insurance practice, including investigation, assessment, and the training of non-medical claims handlers and loss adjusters.

Alongside my insurance career, I have always been interested in using technical approaches to organize complex work. Over the years, I consistently applied structured thinking and simple technical tools to improve daily workflows, decision processes, and management practices, even when these efforts were not formally described as system or software design.

Nearly two years ago, the organisation I work for introduced a team of AI engineers to develop an AI agent for claims-related tasks, and I participated as a domain expert, reviewing AI outputs and discussing implementation approaches with the engineering team.

Early demonstrations were impressive, but real-world use exposed deeper limitations. Claims handling and loss adjusting turned out to be far more complex than expected, and it was clear that simply explaining insurance knowledge to AI engineers and relying on them to design the system was not enough. The project was paused.

On the other hand, after many years working as a domain expert had given me deep expertise, I gradually reached a professional ceiling. The work itself remained complex and demanding, but opportunities for deeper professional development became narrower over time. In addition, handling a large number of serious injury and fatality claims over the years has taken an emotional toll.

All these realisations led me to take over the paused project, continuing to review AI outputs

while also developing prompts and designing system architecture grounded in real insurance practice. Over time, this work led to my current role as an AI Solution Architect. In this article, I share my experience over the past year, showing what it takes to transition from an insurance expert to an AI Solution Architect, including the knowledge to acquire and the work involved.

Chapter 1

I believe that every professional in the insurance industry has built a deep reservoir of expertise through years of dedicated practice. Whether your focus is on property, engineering, marine, or medical cases, your professional background, combined with an understanding of insurance codes, statutes, and mechanisms, makes you a multi-faceted talent by nature.

On a long career path, choices have always existed. Some choose the management route to lead teams; others deepen their technical expertise as specialists; while some enhance their communication and resource integration skills to pursue business development. Now, the emergence of AI offers a new possibility, a technical path centered on domain expertise: participating in, or even leading, the architecture of AI-assisted systems.

Over the past year or more, I have transitioned from an insurance professional to working in AI architecture. During this process, I discovered effective methodologies—and made a few wrong turns as well. What I’ve learned is that in a highly specialized field like insurance, AI architecture must be led by an Insurance expert who has mastered the technical fundamentals, from LLM mechanics to data structuring. By translating our domain expertise into precise data assemblies and signal triggers, we provide the clarity that ensures programmers aren't forced to guess complex industry nuances, but instead, can focus on building a robust, high-performance technical engine. In the following chapters, I will break down the methodologies for building AI Systems, from data structuring to workflow segmentation.

To transition from insurance expert to AI Architect, we must first clarify what today's AI—specifically Large Language Models (LLM)—can and cannot do. Only by understanding these fundamentals can we dive deeper and become true architects of AI Systems that make LLMs truly effective.

In simple terms, mainstream AI today is essentially powered by LLMs built on the Transformer architecture. Their so-called "intelligence" does not come from understanding meaning in the human sense; rather, it is about "**predicting the most probable next word (token) based on all existing inputs.**" Yes, it is fundamentally still a mathematical calculation.

However, this **calculation relies on a "black box"** known as embeddings: high-dimensional vector representations of tokens. While we do not know precisely what each dimension represents, these vectors (commonly 1024, 2048, or even 4096 dimensions) are formed during model training. This is one reason why different models feel noticeably different in terms of their practical utility.

This involves massive computational complexity, driven by factors such as Multi-Head Attention mechanisms, the number of Transformer layers (commonly 64-96 layers), and parameter scale. If we abstract the process, every inference is a continuous cycle: Based on the number of input tokens, N, the model performs an N * N matrix calculation to output the next token. This output token is then added back to the input, and the N * N calculation repeats again and again until the output is complete.

This brings a direct physical limitation. The number of tokens (N) has an upper ceiling. This is what we commonly refer to as the "Context Window" or "Input Limit."

It is important to note that in practice, even if a model supports a massive context window, its **stability and reasoning quality often decline as the input grows**. In long-context scenarios, the model must distribute its attention across more tokens, causing the "attention" to be spread too thin. Key information can easily be "drowned out" by background content, affecting the accuracy of the results.

Furthermore, because LLMs are essentially performing probability calculations in vector

space, they are **NOT naturally suited for deterministic problems** that require exact results. For tasks such as mathematical operations, date judgments, time-sequence analysis, or boundary condition comparisons, the model may provide seemingly plausible answers, but this correctness is unstable and lacks verifiability.

Chapter 2

We now understand the fundamental mechanics of LLMs from Chapter 1. However, before we begin building an AI System, the first task is neither writing prompts nor coding logic. It is organising the raw data.

In actual insurance practice, the information we handle comes from messy, fragmented sources: it includes structured data from Core Systems (e.g., HIS, underwriting platforms, and claims systems), as well as unstructured information like paper documents, photos, audio, and video files. To organise this raw information effectively, there are two critical steps.

The first critical step in the data workflow is **Data Categorisation**. We must clarify which information is critical for AI analysis and which is merely supplementary. This requires high granularity; we cannot simply say, "The policy wording is critical." We must go further, dividing the policy into specific sections and provisions. For instance, we might determine that only "Section A, Provision X" is necessary for a particular task. As we learned in Chapter 1, irrelevant input causes the LLM's "attention" to spread too thin, leading to decreased performance.

The second critical step is **Data Structuring**. Using formats like JSON, which is clear and readable for both humans and machines, we can assemble input data from various sources into a unified structure. Only when structuring is complete can we decide: which content should go to the LLM to leverage its semantic understanding, which should be handled by code for deterministic calculations, and which fields need to be hidden (masked) before being sent to the cloud and restored after the results return.

A typical structured claim record might look like this:

{

"policy_type": "Medical Expense",

"policy_effect_date": "2022-11-02",

"claim_reason": "On the morning of April 1, after sending my child to school, I was riding a three-wheeled vehicle on the way back to my hometown, when suddenly the three-wheeled vehicle was like flying fast, swallowing clouds and riding mist, jumping up and down, the two handlebars of the vehicle swinging back and forth and moving around randomly..... I used all my strength to hold onto the handlebars. At that time, my left hand, arm, shoulder, and the inside of my abdomen were vibrating very severely, after which I fainted and did not know anything anymore. A passerby called an ambulance, which I personally did not know. After examination by the doctors, I was found to have Type 2 diabetes, this being the first time I was diagnosed with diabetes.",

"stated_incident_date": "2023-04-01",

"related_records": [

{

"type": "outpatient",

"hosp": "Hospital A",

"visit_date": "2023-04-01",

"diagnosis": "Syncope",

}

{

"type": "inpatient",

"hosp": "Hospital A",

"admission_date": "2023-04-01",

"discharge_date": "2023-04-06",

"discharge_diagnosis": "Diabetes Type II",

}

]

}

However, simply structuring data is not enough; we must also perform "**Data Refinement**." In this example, the "claim_reason" contains chaotic, metaphor-heavy descriptions ("swallowing clouds and riding mist") that create noise and distract the LLM. More critically, the claimant's subjective assertion, (this being the first time I was diagnosed") can seriously mislead the model's calculation. To ensure an objective analysis, we must refine such messy narratives into concise facts:

"claim_reason": "2023-04-01, Syncope while driving; diagnosed with Type 2 diabetes upon admission."

This refinement prevents the LLM from being swayed by subjective storytelling and forces it to perform a cold, evidence-based comparison between the claim and the medical records.

Compliance is another vital issue. For sensitive data fields, we must determine if they can be de-identified (e.g. using placeholders like "Person A" to replace names and IDs). If a field is critical for analysis but cannot be de-identified, it must be processed on a company's on-premise LLM server. This ensures the data never leaves the corporate boundary. Only de-identified data should ever be considered for cloud-based computing.

During the structuring process, field naming and classification logic directly determine the success of the AI. For example, in medical records, a programmer once used field names like "aux_ex" for auxiliary examinations and "spe_sit" for specialty-specific physical examinations. While the former is clear, the latter is ambiguous and can destabilize the model's vector calculations. When insurance experts with a medical background step in, they recognise that these are simply components of a physical examination and can unify them under a single, clear field like "phy_ex". Experts guide which fields need better naming and which should be excluded from the LLM entirely to prevent errors.

The same applies to fire site inspections. A general photo of a study room is mainly used to estimate renovation areas under Building cover. However, a photo of a pile of books in that same room is used to verify the quantity of personal property under Contents cover. Although the location and the photo batch are the same, the analytical purpose is entirely

different. If a programmer unfamiliar with the business simply classifies them by location, the LLM’s attention will be spread too thin during inference, leading to fundamental errors.

This highlights why an insurance expert is indispensable to AI architecture. Insurance professionals understand business rules and can guide the classification logic to ensure every piece of data is correctly processed. A future AI architect must not only know the business but also master data structuring to ensure this "business common sense" is correctly utilised by the LLM.

When organising image, video, and document data, the best practice is to use different models for what they do best:

1. Multimodal LLMs are particularly helpful for sorting through complex or unstructured visual-text data. For example, they handle irregularly formatted documents, handwritten notes, or photos/videos with mixed content much better than traditional OCR tools. They can quickly grasp the overall context and extract meaningful information from messy sources.
2. However, when the task requires precise measurements, such as calculating the exact area of damage in square meters, counting specific items, or measuring dimensions in photos or videos, multimodal LLMs are far from accurate. For these needs, we turn to specialised vision models like CNNs, which are excellent at delivering reliable numerical results.

In insurance work, combining the strengths of these models gives us the most complete and trustworthy data organisation: multimodal LLMs provide the broad contextual understanding, while vision models supply the precise measurements required for claims assessment.

Chapter 3

After gaining a basic understanding of LLMs and the importance of data structuring, the next step is interacting with the model. Simply dumping structured data into an LLM will rarely yield the precise reasoning or analysis required for insurance work.

This is where we begin to architect an AI System. There are two core operations at this stage:

1. **Prompt engineering**: This acts as the "control panel" for interacting with the LLM. At its core, it influences the model's attention mechanism during computation, amplifying the key dimensions we need while suppressing distracting ones to guide the model toward the desired output.
2. **Process segmentation**: When we ask an LLM to solve multi-layered problems in one go, the output becomes highly unstable (as the "attention" is spread too thin across multiple variables). In such cases, even the best prompts cannot fix the issue. We must use domain expertise to segment the workflow. Additionally, as established in Chapter 1, LLMs are not suited for precise tasks like mathematical calculations or date judgments; these must also be segmented out.

Furthermore, many rule-based SaaS systems have been widely adopted over the years, and their precise judgment capabilities should be maintained. If a rule is already working, do not ask the LLM to handle it. For example, in health insurance underwriting, there are automatic scoring systems based on ICD-10 codes. Using an LLM for the scoring itself would be counterproductive due to its weakness in deterministic calculation. However, using an LLM to extract various diagnoses and ICD-10 codes from massive medical records to fill in gaps is where its true value lies.

Process segmentation must be architected based on the experience of an insurance expert. Take, for example, a case involving a precision lathe destroyed by fire in a factory.

- The steps that can be assigned to AI include: analysing whether the incident falls within the scope of policy coverage, determining if Business Interruption (BI) is triggered, verifying that the policy responds (such as checking exclusion clauses or whether maintenance obligations were met), and conducting a preliminary fraud risk assessment.
- Conversely, some steps are clearly unsuitable for AI and should be handled by code, such as confirming that policy numbers match exactly, determining if the incident date and claim date fall within the policy period, comparing price quotes, and calculating the

specific loss amount for BI.

- Additionally, tasks like interviewing shareholders, site inspection, and negotiations must remain as human-centric Entry Points in the architecture.

In actual construction, these preliminary segments need further granular breakdown. In the lathe fire case, the policy conditions might include a "Maintenance Warranty." Verifying this is a step for AI, but a typical maintenance log contains: *the maintenance date, handwritten notes, and the technician's signature*. For this task, the process should be segmented again:

- A Multimodal LLM is responsible only for information extraction: identifying the date, content and personnel to output structured data. (Note: Personnel names are involved, so this must be handled by a company's on-premises server).
- A code-based tool then calculates the number of days between the maintenance date and the loss date, comparing it against the maintenance frequency required by the policy to flag any anomalies.

Only data analysed through this kind of assembly is compliant, accurate and ready for subsequent LLM to analyse.

In summary, process segmentation should be conducted in layers:

1. Determine if the business steps can be naturally separated or have clear sequential relationships
2. Decide if further segmentation is needed based on the input volume and judgment complexity of each step
3. Perform the deepest layer of decomposition for deterministic calculations and data privacy protection.

Chapter 4

After completing the Process segmentation, we move to the stage of individual LLM interaction: writing prompts. While many tutorials and tips are available online, I prefer a modular approach, which makes it easier to review from a business logic perspective. Here are the modules I frequently use:

1. **Role**: Significantly shifts the LLM's output style and attention direction. Note: Over-embellishing (e.g. "senior authoritative expert") is often ineffective, while "downward simulation" (e.g., "intern claims adjuster") works remarkably well.
2. **Intent & Scope**: Define "what to do" and "what to avoid." Intent should use clear verbs (e.g., "extract," "judge," "compare") rather than vague ones. Scope constrains the boundaries—for example, "Analyse based only on provided records; do not judge based on speculative information."
3. **Input/Context**: Explain exactly what the structured input contains. This is crucial when field names are ambiguous. For example, if there are three dates—date_01, date_02, date_03—you must clarify: date_01 is the "Application Date," date_02 is the "Premium Payment Date," and date_03 is the "Policy Inception Date."
4. **Process**: Break down how an insurance expert thinks into specific steps. For example, in health insurance, analysing a Pre-existing Condition follows a path: "Identify disease name → Is it an acute illness that can recur? → Is it related to the current claim? → Provide conclusion." Avoid useless instructions like "traverse word by word," as Transformers process all tokens simultaneously.
5. **Constraints**: Set clear boundaries based on business needs, such as "Abnormalities found without a final doctor’s diagnosis shall not be treated as a confirmed disease."
6. **Reference**: Embed small amounts of common sense to prevent the model from relying on its own potentially outdated or inaccurate internal knowledge.
7. **Error Handling**: Provide a "safety net" for exceptions. For example, "If the input claim notification is empty or contains unintelligible characters, output: 'No claim information found, please check'."
8. **Output**: Strongly recommend structured formats (like JSON). This is essential for multi-LLM architectures to ensure results can be handled by code or subsequent steps.

Prompt writing is an iterative process. An insurance expert’s advantage lies in better understanding the business logic and identifying output deviations quickly during testing. For extremely simple tasks, a modular structure isn't always necessary; a single-paragraph prompt can work.

Crucially, since different LLM tasks require code for interaction and data transfer, communication with programmers is vital. When leading an AI architecture, you must

manage Interface design and data format. This includes writing API Documentation for each LLM interaction:

- How the input payload is assembled
- The data sources
- The Json structure of the assembly
- The expected output structure.

Chapter 5

For efficiency and accuracy, the golden rule is: **keep the task per prompt as simple as possible**. If a task is too complex, prioritise further segmentation rather than writing massive, complicated instructions.

This is especially true in medical-related health insurance. For instance, after extracting the history of a diagnosis, the model needs to judge if it meets the policy coverage. If you feed all Policy Provisions (Medical Reimbursement and Critical Illness) to the LLM at once, the results may become unstable. Because these provisions share many common clauses, the few differentiating clauses might not trigger enough "attention" change in the model's vector calculations.

In this case, it is better to split the Policy Provisions into two separate tasks for two different LLM calls. If the "Critical Illness" judgment remains unstable because "Mild," "Moderate," and "Severe" conditions share 95% of the same text, it is more effective to split these into three independent LLM tasks rather than endlessly tweaking the prompt.

Some might ask: Why not use RAG? I have been down that road, and for Policy Provisions that are 95% identical, RAG recall is often poor. To achieve better results with RAG, you would need to pre-process the provisions into a structured format with rich metadata tags (e.g., tagging a specific clause as "Mild Condition"). However, once the provisions have been organized into such a structured format, having the code directly call the specific clause becomes both more accurate and efficient than using RAG. This also implies the need for a separate pre-processing AI System to structure and tag all Policy Provisions

before the main AI System starts its work.

Regarding model deployment: Insurance is a rigorous industry. Unless the task is very simple, I recommend models with at least 30B+ parameters. Set a low Temperature (0 to 0.1) and an appropriate TopP (0.6 to 0.8). **Disable the model's "thinking" and "web search" capabilities** to prevent uncontrollable variables and improve stability.

When an insurance expert masters process segmentation, AI architecture and prompt design, they can directly lead the construction of AI Systems. In this dimension, it is very difficult for programmers to replace them.

Chapter 6

When an insurance expert has enough LLM knowledge to lead an AI project, they must realise that it involves more than just Process Segmentation, prompt writing, and coding. You must recognise that success requires close coordination with several key roles:

1. **Programmers (Backend and Frontend):**
   Backend developers handle database integrations, data storage, and logging of all AI System records for traceability, audit, and future analysis. Frontend developers connect the system to B2B or consumer-facing apps, ensuring seamless user interaction and data flow.
2. **Operations (Ops):**
   Server configuration and network conditions limit compute and concurrency. Ops also set up UAT (User Acceptance Testing) environments and manage deployment pipelines. Early confirmation of parallel processing capabilities is essential to avoid rework.
3. **Data Security Experts**:
   Determining which data can leave the internal network and which must remain on on-premises servers is a core constraint. Early alignment prevents costly re-segmentation of tasks and ensures compliance.
4. **Test Engineers:**
   LLM projects cannot rely on end-of-process testing alone. Without deep understanding

of business logic, testers can only verify format, not substantive accuracy.

I once led an AI System project where everything seemed perfect during testing. However, after launch, I noticed a slight drift in the results. The cause was not the model or the code, but data assembly: the programmer had accidentally included "pre-policy diagnoses" into the "post-policy diagnosis set."

To a programmer, this was just a minor field-joining error. To a tester without a medical background, it was nearly impossible to catch. In my prompt design, that data set was only a secondary reference, so the error wasn't obvious early on. This wasn't a "bug" or "hallucination"; it was Semantic Drift in the input. When this drift is passed through subsequent tasks, it amplifies and ruins the consistency of the final judgment.

Therefore, every role must understand the overall design early. Even for the insurance expert leading the architecture, it is beneficial to introduce Cross-reviews with other insurance experts at key nodes to enhance system stability.

Chapter 7

By understanding the essence of LLMs, data organisation, process segmentation, prompt design and cross-role communication, you have mastered the core methods of an AI Architect.

First, the core capability of an AI architect in the insurance industry is not fine-tuning model details, but converting professional knowledge into systematic thinking. This is why AI built solely by programmers struggles with core insurance practice like underwriting and claims: the entry barrier is too high. Many projects currently labeled "AI Underwriting/Claims" are actually still based on traditional RPA, OCR and rule engines, with LLMs used only for peripheral support. The core judgment still relies on preset rules or humans.

A true insurance expert-turned-AI architect knows how to divide modules, set boundaries, manage dependencies and decide what goes to the LLM, what stays with deterministic calculation, and where the human entry points must be. This is the core competitiveness of

the final product.

Second, the advantage of a domain-led architecture is the ability to detect subtle semantic deviations that programmers or testers might miss. This is the foundation of a reliable, sustainable AI system.

Finally, while managing the project, the AI Architect must ensure every step is controllable and traceable, from the programmer’s data format conversion to the Ops team’s concurrency management and the security team’s data audit.

In summary, becoming an AI architect is not about mastering the latest algorithms, training a model, or writing complex code. It is about transforming your professional expertise into a detailed, systematic logic.

In the near future, InsurTech code will be cheap and computing will be a commodity. **The true core competitiveness of an AI system will lie in its architectural design; specifically, how an expert dismantles complex data, determines the sequence of processing, and orchestrates the re-assembly of information to reach the desired outcome.** This deep understanding and precise dismantling of business logic is the true "soul" of the architecture, and it remains irreplaceable. This is the home ground for every senior insurance expert.

Glossary of Key Terms

- **AI System**: An autonomous system designed to use tools, multiple LLMs, and traditional code to make decisions and complete complex insurance tasks.
- **LLM (Large Language Model)**: AI systems like GPT-4 that are trained on massive datasets to understand and generate human language.
- **Transformer**: The foundational neural network architecture that enables modern AI to process sequences of data efficiently.
- **Token**: The basic unit of text that AI processes. One token is roughly equivalent to 0.75 of an English word.
- **Embedding**: The process (and result) of converting tokens into high-dimensional

vectors so that the model can represent and calculate the "closeness" of meanings.

- **Inference**: The stage where a trained AI model produces an output or prediction based on a given input.
- **Deterministic**: Logic or calculations that have a fixed, certain answer based on rules (e.g., 200 + 40 = 240), as opposed to probabilistic AI predictions.
- **Context Window**: The maximum amount of text (tokens) a model can "remember" and process in a single interaction.
- **HIS (Hospital Information System)**: A comprehensive information system designed to manage all aspects of a hospital's operation. In insurance, it is a primary source of medical evidence.
- **Data Categorization**: The process of sorting data into relevant groups to ensure the AI only processes information necessary for the specific task.
- **Data Structuring**: The process of converting "unorganized" information (like a paragraph of text) into a "fixed" format (like a JSON table) so that computers can easily analyze it.
- **JSON (JavaScript Object Notation)**: A standard text-based format for representing structured data, widely used to transmit data between systems.
- **OCR (Optical Character Recognition)**: Traditional technology for extracting text from images or scanned documents; it performs less well on complex layouts or handwriting compared to multimodal LLMs.
- **Multimodal LLM**: An AI model that can process and understand multiple types of data together (text, images, videos, handwritten notes). It is especially effective for complex, irregularly formatted, or handwritten documents where traditional OCR often struggles.
- **CNN (Convolutional Neural Network)**: A specialized AI model for analyzing images and videos. It excels at precise tasks such as measuring areas, counting objects, or detecting specific features accurately.
- **De-identification**: The process of removing or masking personal identifiers so that the remaining information cannot be traced back to an individual.
- **On-premises**: Refers to software or servers that are installed and run on computers on the premises of the organization, rather than in the cloud.
- **Prompt Engineering**: The practice of refining inputs to an LLM to get more accurate, high-quality, or specific results.
- **Process Segmentation**: The method of breaking down a complex business workflow into smaller, manageable tasks that can be handled by either AI or traditional code.

- **ICD-10 (International Classification of Diseases, 10th Revision)**: A globally recognized system for coding medical diagnoses and procedures.
- **Business Interruption (BI)**: Insurance coverage that replaces business income lost in a disaster.
- **Shareholders**: In insurance claims context, this term refers to parties with an insurable interest in the loss or policy (e.g., property owners, directors, partners, mortgagees) who may need to be interviewed.
- **Entry Points**: Pre-defined stages in an AI workflow where humans must intervene or where external systems can be called.
- **Policy Responds**: The determination of whether a policy's terms and conditions apply to a specific claim or circumstance.
- **Interface**: The shared boundary or "bridge" where two separate systems (like an LLM and a database) exchange information.
- **API Documentation**: A technical guide that explains how to use an interface, including what data to send and what format to expect back.
- **RAG (Retrieval-Augmented Generation)**: A technique that gives an LLM access to external data to improve its answers, though it can struggle with highly similar documents.
- **Temperature**: A parameter that controls the "randomness" of an LLM's output. Lower values make the output more focused and deterministic.
- **TopP**: A parameter that limits the model's word choices to a subset of the most likely tokens, helping to balance diversity and accuracy.
- **30B Parameters**: Refers to the "size" of the AI model. Generally, models with 30 billion parameters or more have stronger reasoning capabilities for complex tasks.
- **Compute**: The processing power (CPU/GPU) required to run AI models.
- **Concurrency**: The ability of a system to handle multiple tasks or LLM calls at the same time.
- **Semantic Drift**: A phenomenon where the meaning or context of data shifts slightly during processing, leading to unintended outcomes in AI reasoning.
- **Cross-review**: A quality control process where another expert in the same field reviews the logic or output to identify errors.
- **Systematic Thinking**: The ability to understand how different parts of a business process influence each other within a whole system.
- **RPA (Robotic Process Automation)**: Software "robots" that automate repetitive, rule-

based tasks (like data entry) but lack human-like reasoning.

- **Deterministic Calculation**: Logic or calculations that have a fixed, certain answer based on rules, as opposed to probabilistic AI predictions.

About the Author

Hongxiang Sun is an AI Solution Architect and Senior Loss Adjuster with over 20 years of experience in insurance claims and risk assessment. He has a background in clinical medicine and in recent years has focused on architecting AI-enabled claims workflows grounded in real-world insurance practice.