Satya Sai Mudigonda, CPCU, PMP, AIAI |
Professor

Rohan Yashraj Gupta, PhD, FIA, FIAI

Agentic AI for Actuaries

First Edition

ACTEX Learning

Request for permission should be addressed to:
ACTEX Learning
PO Box 69
Greenland, NH 03840
support@actexlearning.com

Manufactured in the United States of America

Cover Design by Rachel Maragnano

Publisher's Note

For over 50 years, ACTEX Learning has had the privilege of publishing for a profession that approaches change with care and foresight. That care is well founded, as actuarial work carries statutory weight and professional accountability, and its methods rightly earn their place over time rather than by novelty. It is in that spirit that we are pleased to introduce a foundational text on a subject that will shape the profession significantly in the future, and to share a few words about why we believe it deserves a place in every actuary's library.

Artificial intelligence has already begun to shape actuarial work. Across insurers and consultancies, we have watched a steady progression from early curiosity to pilot projects, and increasingly to systems that support pricing, underwriting, claims, reserving, and risk management. Alongside that progress, we have heard a consistent request from students, practitioners, and employers alike for practical guidance on how to use these tools with care – today and in the future. Specifically, how to build, evaluate, and govern such systems so that the work they support meets the standards of reliability and documentation the profession requires.

Satya Sai Mudigonda and Rohan Yashraj Gupta approach this task from a perspective we find persuasive. Actuaries need not become artificial intelligence engineers. What matters is understanding these systems well enough to exercise judgment about them: what an agent can reasonably be trusted to do, where it is likely to fall short, what evidence would show that it is working as intended, and which decisions should always remain with a named individual. This is a distinct skill, and one that draws naturally on strengths the profession already cultivates, including mathematical maturity, professional skepticism, and the discipline of documenting judgment.

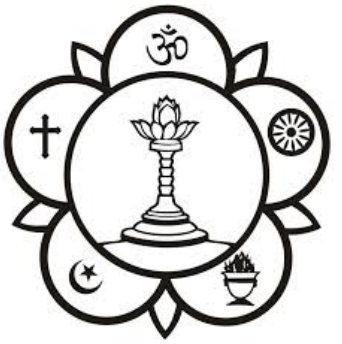
The authors also believe that this material is best learned by building, and our experience supports this approach. Agentic AI can feel abstract in description and becomes considerably clearer once a reader has written an agent, connected it to a tool, and followed its reasoning through a task from beginning to end. This is therefore a working textbook. The code is written to be run and the case studies to be adapted, and the authors provide a companion repository and website so that readers may begin without the preliminaries of setting up a development environment.

Two practical observations may be helpful. The first is that AI moves more quickly than publishing schedules allow. The architectural ideas presented here should long outlast the particular frameworks, tools, and model names used to illustrate them, and the authors have written their explanations accordingly. Readers working with a specific framework should expect to consult its current documentation alongside these pages, as interfaces and capabilities continue to evolve. The second is that this book is intended as an educational resource, and does not constitute actuarial, legal, or regulatory advice. The systems described here are aids to actuarial work and not substitutes for professional responsibility, which continues to rest with the qualified actuary. The accompanying datasets are synthetic and provided for learning purposes only.

Our thanks go to Satya Sai Mudigonda and Rohan Yashraj Gupta for entrusting this work to us, and to the many students and practitioners whose thoughtful review helped shape it. We hope readers find it a useful resource, and we welcome corrections, questions, and suggestions for future editions.

Best Regards, Bill Marella CEO bill@actexlearning.com

Dedication



'Love All, Serve All.'

'Help Ever, Hurt Never.'

With deepest reverence and gratitude,
we dedicate this book, Agentic AI for Actuaries, to

Bhagawan Sri Sathya Sai Baba

whose divine blessings, wisdom, and teachings continue to guide humanity
towards compassion, selfless service, righteousness, and purposeful living.

May this work contribute, in its own humble way,
towards the ethical and responsible application of knowledge
for the welfare of society.

Foreword

The actuarial profession has always stood at the intersection of business, economics, finance, risk management, societal responsibility, statistics, data science, computing, analytics, artificial intelligence, machine learning, and mathematics. Over the years, it has evolved into a highly multidisciplinary field that integrates technological innovation with analytical thinking to address increasingly complex real-world challenges.

Modern actuaries are no longer confined to traditional insurance and pension domains alone. They operate in a rapidly transforming environment shaped by big data, digital transformation, climate risk, healthcare innovation, cyber risk, and global financial uncertainty.

Today's actuaries work extensively with programming languages, predictive modelling, cloud computing, AI-driven systems, and advanced analytical frameworks to evaluate uncertainty and support informed decision-making. They collaborate closely with data scientists, software engineers, economists, financial analysts, regulators, and business leaders to design sustainable, resilient, and socially responsible solutions.

Today, we stand at the threshold of another transformational shift: the rise of Agentic Artificial Intelligence. Agentic AI for Actuaries arrives at a defining moment for the profession. Artificial Intelligence is no longer limited to automation or data analysis alone. Modern AI systems are increasingly capable of reasoning, planning, collaborating, learning from feedback, and autonomously executing complex tasks. These 'agentic' capabilities have the potential to fundamentally reshape actuarial work across pricing, reserving, underwriting, claims analytics, enterprise risk management, capital modelling, customer engagement, and strategic decision-making.

Yet, technology alone does not define progress. The actuarial profession is built upon principles of trust, ethics, accountability, and service to society. As AI systems become more powerful, the responsibility of actuaries becomes even greater. The profession must not only adopt these technologies but also guide their ethical, transparent, and responsible application. Human judgement, professional scepticism, governance, and values will remain central to actuarial practice.

This book provides a timely exploration of how Agentic AI can augment actuarial capabilities while preserving the core principles of the profession. It bridges the gap between emerging AI concepts and practical actuarial applications in a manner that is accessible to students, professionals, educators, and industry leaders alike. The authors thoughtfully combine technological understanding with professional context, helping readers appreciate both the opportunities and the challenges ahead.

Importantly, this work encourages actuaries to move beyond viewing AI merely as a tool. Instead, it invites the profession to reimagine workflows, collaboration, learning, and decision-making in an AI-enabled future. Those who embrace continuous learning and adaptability will be best positioned to lead the next generation of actuarial innovation.

We congratulate the authors for their vision and contribution in bringing together the worlds of actuarial science and Agentic AI in a meaningful and forward-looking manner. We hope this book inspires readers to explore, innovate, and lead responsibly in the evolving landscape of intelligent systems.

May this work contribute positively to the advancement of the actuarial profession and to the broader goal of using technology for the welfare of society.

Sri Sathya Sai Institute of Actuaries (SSSIA)

Acknowledgements

‘Hands that help are holier than lips that pray.’
— Bhagawan Sri Sathya Sai Baba

This book has grown out of a journey of service, learning, collaboration, and a shared belief in the possibilities that emerge when actuarial science meets new and evolving technologies. Above all, we offer our humble gratitude to Bhagawan Sri Sathya Sai Baba for His divine inspiration, guidance, and blessings, which have been a constant source of strength throughout this journey.

We are sincerely grateful to the 27+ core members of the Sri Sathya Sai Institute of Actuaries for their continued support, encouragement, and commitment to serving the actuarial community. For the past 16 years, the Institute has worked to make actuarial education, mentoring, training, workshops, and student support available completely free of charge, as a form of selfless service to society and the profession. The dedication and collective effort of its members have allowed this mission to grow steadily over the years, and we hope to continue carrying it forward with the same spirit and commitment for many years to come.

We are also deeply thankful to the 1,200+ members from more than 180 institutions in India and abroad who are part of this growing community. Many members reviewed sections of the manuscript and shared their thoughts, perspectives, and feedback. Their curiosity, enthusiasm, and willingness to engage with new ideas added real value to this work and strengthened our belief that the next generation of actuaries should combine strong actuarial foundations with an understanding of emerging technologies such as Agentic AI.

A major influence on this book has been our experience conducting Agentic AI for Actuaries workshops and related learning initiatives. Across 7 workshops, more than 250 employees from 27+ organisations have been trained in the practical application of Agentic AI in an actuarial context. These initiatives have also led to 54 student projects and 36 employee case-study projects, giving participants the opportunity to move beyond theory and apply these concepts to real actuarial and business problems. The questions raised, ideas explored, solutions developed, and lessons learned through these projects have shaped many of the practical perspectives reflected in this book.

On a personal note, we are deeply grateful to our parents and family members for their constant encouragement, support, patience, and belief in us. Their values and guidance have been a source of strength throughout this journey and have helped us continue our educational and service efforts with purpose and dedication.

As we bring forward this work, we do so with a sense of gratitude and responsibility, and with the hope that the ideas and experiences shared in this book will contribute meaningfully to the continuing evolution of the actuarial profession.

We sincerely thank ACTEX Learning for publishing this book.

With deep gratitude,
Satya Sai Mudigonda and Rohan Yashraj Gupta

Preface

This book is for actuaries who want to understand agentic AI without first becoming software engineers, and without wading through evangelism. It is a complete treatment: eighteen chapters spanning the foundations of artificial intelligence through to production deployment, with hands-on Python code, extended case studies across pricing, reserving, life and health, pensions, and risk, and chapter-length attention to governance. It is written for those who intend not merely to understand these systems, but to build, evaluate, and be accountable for them.

We have written for three audiences simultaneously: aspiring actuaries who will graduate into a profession that already uses these tools, practising actuaries in pricing, reserving, life, health, pensions, and risk who must evaluate AI systems for production use, and actuarial leaders who must decide what to adopt, what to defer, and what to refuse. We assume strong mathematical maturity and fellowship-level actuarial vocabulary. We do not assume any prior exposure to artificial intelligence, machine learning, or programming. The code is written to be read: every Python block is explained line by line, and a reader who has never opened an interpreter can follow the argument, while a reader who has can reproduce it.

The structure is deliberate. Part I builds the foundations of artificial intelligence and machine learning. Part II covers large language models and how to work with them. Part III introduces agentic AI proper, the architectures that distinguish autonomous systems from chatbots. Part IV applies these techniques to the four major actuarial domains, each grounded in a running case study developed to working depth. Part V addresses deployment, governance, and what comes next. Each chapter opens with a short vignette grounding the central concept in a concrete actuarial situation, develops the argument in plain prose alongside executable code, and closes with a summary, reader takeaways, and direc-

ted further reading. The book can be read end-to-end, but it is also built to be worked through: the case studies and code are designed to be run, modified, and extended against the reader's own problems.

Two cautions are worth stating at the outset. AI does not replace the qualified actuary. The Appointed Actuary, the Actuarial Function Holder under Solvency II, the signing actuary on a Statement of Actuarial Opinion, all bear personal professional liability that no algorithm can carry. The structural argument for partnership, not displacement, is developed in Chapter 1 and assumed thereafter. And the field is moving quickly: specific tools and frameworks named in this book will date faster than the architectural ideas behind them. Where we show code, we have chosen frameworks for their pedagogical clarity as much as their current standing, and we have weighted the prose toward the ideas that will outlast them.

Satya Sai Mudigonda, CPCU, PMP, AIAI and
Dr Rohan Yashraj Gupta, FIA, FIAI
Sri Sathya Sai Institute of Actuaries (SSSIA)

Table of Contents

How to Use This Book

This book is designed to be read in sequence, as each part builds on the ideas and terminology introduced earlier. Most of the case studies follow Meridian Re, a fictional composite reinsurer that is used throughout the book to make the concepts more practical and easier to connect.

Parts I and II (Chapters 1 to 8) are completely code-free. If your main interest is in understanding the concepts rather than writing or running programs, you can continue reading the explanations and case studies in Parts III to V and simply skip the code listings.

The coding begins in Chapter 9. Every code example from Chapters 9 to 17 is available as a runnable Python script in the companion repository:

`https://github.com/RohanYashraj/agentic-ai-for-actuaries-code`

If you would prefer to try the examples without installing anything locally, you can use the companion website:

`https://aiforactuaries.sssia.org`

The tool scripts run directly in your browser, while the agent-based examples run live. Each chapter is also available as a Google Colab notebook for readers who prefer a notebook-based environment.

To run the Colab notebooks or the code locally, you will need a Google AI Studio API key. The free tier is sufficient for all the examples included in this book.

All datasets included in the repository are synthetic and are intended only for learning and demonstration. They should not be used for real actuarial or business work.

In a few places, the repository may differ slightly from the code shown in the printed book. Where this happens, the relevant chapter README explains the reason for the difference. Since the repository can be updated over time, it should be treated as the most current version of the code.

Part II

Working with Large Language Models

5

Prompt Engineering for Actuarial Professionals

Code in this chapter. The listings in this chapter are printed to be read, not copied. Frameworks and provider APIs move faster than a printed book can, so the maintained and tested version of every listing lives in the companion repository at `https://github.com/RohanYashraj/agentic-ai-for-actuaries-code`, together with pinned dependency versions and any corrections issued after publication. Where the repository and the printed listing differ, the repository is current.

The reserving actuary at a US commercial lines carrier had until Friday to deliver the narrative section of the Statement of Actuarial Opinion. The Schedule P exhibits were finished. The reserve ranges were signed off internally. What remained was the commentary — three pages explaining why workers compensation development had accelerated in accident years 2021 and 2022, why the general liability line needed a stronger risk margin, and why her opinion still sat within the range of reasonable estimates. She opened a chat window with a Large Language Model (LLM), typed a short request for a first draft, and received a confident, fluent paragraph that got the direction of the 2022 workers compensation trend wrong.

The mistake took her an hour to find because the prose around it sounded correct. The problem was not that the model was incapable of the task. The problem was that the question she had asked was not the question she had meant to ask.

5.1 Why Prompt Engineering Is an Actuarial Discipline

Chapter 4 established that Large Language Models fail in three characteristic ways for actuarial work: fabricated citations, confident arithmetic errors, and incorrect interpretation of regulation or actuarial method. It also established the mitigation hierarchy — from weakest to strongest, plain prompting, few-shot prompting, Retrieval-Augmented Generation (RAG), tool use, and human review at the point of signature. This chapter covers the first two levels of that hierarchy. It is the first layer of engineering discipline available to an actuary, and for a surprising range of tasks it is sufficient.

The chapter is deliberately written for actuaries using LLMs through web chat interfaces — Claude.ai, Gemini, ChatGPT, and equivalents — rather than through an API. This is where most actuarial use of LLMs begins, and where the first productivity gains typically accrue. The techniques in this chapter work identically across the major web chat tools; the examples use one mainstream chat interface as the reference, but the same prompts produce comparable output in the other major interfaces with minor variations in tone and length.

Prompt engineering is neither a magic art nor a passing fad. It is the practical discipline of specifying a task to an LLM with enough precision that the output is usable, auditable, and — when it fails — wrong in ways that are detectable. That last qualifier matters more than any technique in this chapter. An LLM that produces a plausibly wrong answer is a professional hazard; an LLM that produces a clearly wrong answer is merely a tool that failed a check.

A prompt is a specification. The more of the problem an actuary can pin down in the prompt — the role, the context, the exact output format, the constraints, the examples of what acceptable looks like — the less the model has to guess. Models are trained to produce fluent text. When under-specified, they produce fluent text that drifts toward the most common pattern in their training data, which is rarely the exact pattern a specific actuarial task needs.

ACTUARIAL PARALLEL: A Prompt is a Specification to a Junior Actuary

An experienced actuary who asks a junior colleague to 'draft the reserving commentary' will get one kind of output. The same actuary who specifies the audience (the board risk committee), the structure (three paragraphs covering development, assumption changes, and range rationale), the data cut-off (as at 30 September), the tone (cautious, non-technical), and the length (400 words) will get a materially different, and more useful, output.

Prompting an LLM works the same way. The ambiguity that a junior colleague would resolve by asking a clarifying question is ambiguity the model resolves by guessing. The actuary who writes the clearer specification gets the better draft.

The uncomfortable implication is that prompt quality is a form of professional skill. It is not a substitute for actuarial judgement, but it is adjacent to it. An actuary who can decompose a complex task into unambiguous sub-tasks is already doing half the work of good prompt engineering. The remaining half is learning the patterns in this chapter and knowing when each one fails.

5.2 Anatomy of a Good Prompt

A well-constructed prompt for an actuarial task has six components. None of them is strictly required, and short prompts for trivial tasks can omit several. But for any task where the output will be reviewed, filed, or relied on, all six earn their place.

The first is role. Telling the model that it is acting as a reserving actuary, a pricing consultant, or a regulatory compliance reviewer measurably shifts the register and vocabulary of the output. This is not because the model has a self-concept; it is because role language is a strong signal in the training data about what kind of text should follow.

The second is context. What line of business, what jurisdiction, what accounting framework, what reporting period. A prompt that asks for commentary on IFRS 17 produces different output from a prompt that asks for commentary on IFRS 17 as applied to a US-domiciled commercial lines carrier transitioning from NAIC Statutory Accounting Principles (SAP). The latter is a narrower and more useful request.

The third is the instruction itself — what the model is being asked to do. This should be one clear task, not a bundle. 'Summarise the loss development and recommend a risk margin and draft the SAO narrative' is three tasks. The model will do all three, but each one worse than if asked separately. Prompt chaining, covered in Section 5.6, is the discipline of breaking such bundles into sequential single-task prompts.

The fourth is format. If the output needs to be a bulleted list, say so. If it needs to be a structured table, specify the columns. If it needs to be a 250-word paragraph, give the word count. Models produce text of whatever length and shape is most statistically likely for the prompt; the actuary shapes the output by shaping the request.

The fifth is constraints. What the model must not do — not speculate about unstated assumptions, not fabricate citations, not include any loss ratio estimates, not refer to named competitors. Constraints are easier for models to follow than freeform instructions because they are binary: either the output violates them or it does not.

The sixth is examples. One to five examples of the desired input-output pattern, placed before the actual request. This is few-shot prompting, and it is the second-strongest prompt engineering technique in the hierarchy — covered in its own right in Section 5.3.

The contribution of each component becomes visible in a three-stage refinement. Consider the task of summarising a commercial property claim description into a file note. A weak prompt contains only the instruction. A better prompt adds role and context. A production-quality prompt adds format, constraints, and an example.

STAGE 1 — Weak prompt (instruction only)

Summarise this claim: Insured reports water damage following overnight pipe burst in second-floor server room. Building HVAC logs show temperature drop from 21C to -2C between 02:00 and 04:00. Initial adjuster estimate USD 340,000 for structural and equipment damage.

The weak prompt produces a summary that is technically correct but uneven in register — sometimes conversational, sometimes technical, usually with filler phrases that an actuary would strip out before the note went into a file.

STAGE 2 — Better prompt (role and context added)

You are a property and casualty claims analyst working at a US commercial lines carrier. Summarise the following claim description into a short file note suitable for inclusion in a quarterly reserving review.

CLAIM DESCRIPTION: Insured reports water damage following overnight pipe burst in second-floor server room. Building HVAC logs show temperature drop from 21C to -2C between 02:00 and 04:00. Initial adjuster estimate USD 340,000 for structural and equipment damage.

Adding role and context narrows the register. The output is now recognisably a reserving file note rather than generic summarisation. It still tends to be too long — a paragraph when two sentences would suffice — and it sometimes invents context that was not supplied, such as the implied cause of the pipe burst.

STAGE 3 — Production-quality prompt (format, constraints, example added)

You are a property and casualty claims analyst working at a US commercial lines carrier. Summarise the following claim description into a short file note.

FORMAT: Three sentences maximum. No bullet points. No headings.

CONSTRAINTS: Use only the facts in the claim description. Do not speculate on the cause of loss, the extent of coverage, or the likely reserve adequacy. Do not include opening or closing pleasantries.

EXAMPLE:

INPUT: Fire damage at insured restaurant, kitchen grease fire reported at 23:00, fire services in attendance, adjuster estimate USD 120,000.

OUTPUT: On the reported date, a grease fire originated in the kitchen of the insured restaurant at approximately 23:00 and was attended by fire services. The adjuster's initial estimate of the loss is USD 120,000. No further detail on coverage or cause is available in the current file.

CLAIM DESCRIPTION: Insured reports water damage following overnight pipe burst in second-floor server room. Building HVAC logs show temperature drop from 21C to -2C between 02:00 and 04:00. Initial adjuster estimate USD 340,000 for structural and equipment damage.

The Stage 3 prompt produces a three-sentence note that mirrors the example structure, confines itself to supplied facts, and reads like the file notes the reserving team already produces. The prompt is longer to write once, but it is reusable — the same scaffold works for every claim in a batch by swapping only the claim description.

5.3 Prompting Techniques

Five techniques dominate the current practice of prompt engineering: zero-shot, few-shot, chain-of-thought, self-consistency, and tree-of-thought. Each addresses a specific failure mode, and each has a corresponding set of tasks where it earns its complexity. An actuary does not need every technique for every task; what is needed is the judgement to pick the right one.

5.3.1 Zero-Shot Prompting

Zero-shot prompting is what most users do by default: ask the model to perform a task without supplying any examples. The model relies entirely on its pre-trained knowledge to interpret the request. For well-defined tasks where the model has seen many similar examples in training — summarising a document, classifying a claim description, generating a plain-language paraphrase of a policy clause — zero-shot prompting works reliably.

The technique fails when the task has a specific format the model has not seen, or when the domain vocabulary is specialised enough that the model defaults to generic phrasing. A zero-shot prompt asking for 'an SAO narrative paragraph' will produce a narrative paragraph, but the tone, structure, and technical specificity will reflect the average SAO narrative in the training corpus rather than the specific style of the filer.

Zero-shot is the right choice when the task is generic enough that the default pattern matches what the actuary needs. Ask for a plain-language explanation of Chain Ladder, a glossary of IFRS 17 terms, or a summary of a published research paper, and zero-shot will work. Ask for anything that needs to match a house style or a specific jurisdictional framing, and zero-shot will not be enough.

5.3.2 Few-Shot Prompting

Few-shot prompting supplies one or more worked examples of the desired input-output pattern before the actual request. The model, trained to continue patterns, matches the style, structure, and level of detail demonstrated. Few-shot prompting is the single most reliable way to get an LLM to match a house style, adopt a specific output schema, or reproduce a particular level of technical precision.

For actuarial work, few-shot prompting is particularly valuable for regulatory filings, peer review comments, and standardised narrative outputs — anywhere the shape of the output matters as much as the content. Two

to five examples is the typical range; beyond five, the return on additional examples diminishes, and the prompt grows unwieldy in the chat window.

The refinement pattern for few-shot is to start weak, then add examples until the output matches the house style. The task below is drafting a Schedule P loss development commentary paragraph for a US commercial lines carrier.

STAGE 1 — Zero-shot baseline

Draft a commentary paragraph explaining the 2022 accident year workers compensation paid loss development factor of 1.41 at 12-24 months, which exceeds the preceding three-year average of 1.22.

The Stage 1 output will be a competent paragraph that reads like a textbook explanation of loss development acceleration. It will use generic terminology, will offer confident speculation about causes (rising medical inflation, changes in claim handling, shifts in claimant behaviour) that was not supplied in the prompt, and will not match the carrier's filing style.

STAGE 2 — Few-shot with one example

You are a reserving actuary drafting Schedule P commentary for a US commercial lines Statement of Actuarial Opinion. Match the style of the example below.

EXAMPLE:

INPUT: GL 2020 AY paid LDF 12-24 months: 1.62 (prior three-year average: 1.48).

OUTPUT: The 2020 accident year general liability paid loss development factor of 1.62 at 12-24 months exceeds the preceding three-year average of 1.48. The acceleration is consistent with the reopening of bodily injury claims deferred during the 2020 court closures and has been reflected in the selected ultimate by applying additional weight to the Bornhuetter-Ferguson method over Chain Ladder for this accident year.

NOW DRAFT THE EQUIVALENT FOR:

INPUT: WC 2022 AY paid LDF 12-24 months: 1.41 (prior three-year average: 1.22).

Stage 2 is a substantial improvement. The output matches the sentence structure of the example, adopts the same technical register, and closes with a reserving-method reference. But with only one example, the model still tends to invent causes — in this case, it will often suggest medical inflation or claim-handling changes even though no such information was supplied.

STAGE 3 — Few-shot with two examples and a closed-world constraint

You are a reserving actuary drafting Schedule P commentary for a US commercial lines Statement of Actuarial Opinion. Match the style of the two examples below. Use only the figures in the INPUT. Do not speculate on causes not stated.

EXAMPLE 1:

INPUT: GL 2020 AY paid LDF 12-24 months: 1.62 (prior three-year average: 1.48).

OUTPUT: The 2020 accident year general liability paid loss development factor of 1.62 at 12-24 months exceeds the preceding three-year average of 1.48. The acceleration has been reflected in the selected ultimate by applying additional weight to the Bornhuetter-Ferguson method over Chain Ladder for this accident year.

EXAMPLE 2:

INPUT: WC 2021 AY incurred LDF 24-36 months: 1.08 (prior three-year average: 1.03).

OUTPUT: The 2021 accident year workers compensation incurred loss development factor of 1.08 at 24-36 months is modestly above the preceding three-year average of 1.03. The deviation falls within the historical range of random variation and has not triggered a change in selected development pattern.

NOW DRAFT THE EQUIVALENT FOR:

INPUT: WC 2022 AY paid LDF 12-24 months: 1.41 (prior three-year average: 1.22).

The Stage 3 prompt adds a second example (which teaches the model to recognise both acceleration and modest deviation as valid patterns) and the closed-world constraint. The output now describes the observed deviation and its reserving implication without manufacturing causes that the actuary has not signed off on. A draft produced this way still requires review — but the review is about the reserving judgement, not about removing hallucinated explanations.

ACTUARIAL PARALLEL: Few-Shot Examples Are Credibility-Weighted Experience

Chapter 4 established that a pre-trained Large Language Model is to an industry mortality table what fine-tuning is to credibility-weighted company experience. Few-shot prompting sits in the same family of ideas, but operates at a smaller scale and at a different moment.

Each example in a few-shot prompt is a miniature piece of company-specific experience supplied at inference time. Five examples will not override what the model learned from trillions of training tokens — the credibility is low — but they are enough to steer the model's output toward the local pattern without any retraining. In credibility terms: the examples have enough Z to shift the mean on the specific dimensions they demonstrate, but not enough to change the model's underlying assumptions about language. This is why few-shot prompting works well for format matching and house-style adaptation, and works poorly for teaching the model new domain facts.

5.3.3 Chain-Of-Thought Prompting

Chain-of-thought (CoT) prompting instructs the model to produce its intermediate reasoning before committing to a final answer. Introduced formally by Wei et al. in 2022, the technique was shown to substantially improve performance on arithmetic, commonsense, and symbolic reasoning benchmarks at the largest model scales. For actuarial work, its most direct application is to the second category of hallucination from Chapter 4 — confident arithmetic errors. A model that is forced to show its working is a model whose errors become visible and therefore catchable.

The simplest form of the technique is to append 'Let's think step by step' to the prompt, or — more usefully for structured tasks — to specify the reasoning structure explicitly. The refinement below asks the model to classify which claims in a batch should be referred to large-loss review.

STAGE 1 — Zero-shot (direct question)

For each of the following claims, tell me whether it should be referred to large-loss review. Our thresholds: refer if incurred exceeds USD 250,000, or if incurred exceeds USD 100,000 and the claim is open more than 365 days.

CLAIM A: Incurred USD 280,000, open 120 days.

CLAIM B: Incurred USD 150,000, open 410 days.

CLAIM C: Incurred USD 95,000, open 500 days.

CLAIM D: Incurred USD 220,000, open 200 days.

Stage 1 produces a list of yes/no answers. The model usually gets them right, but occasionally slips — most commonly on Claim D, where the incurred amount is just below the primary threshold and the reasoning is implicit. Without the working shown, a mistake is invisible to the reviewer.

STAGE 2 — Explicit chain-of-thought instruction

For each of the following claims, tell me whether it should be referred to large-loss review. Our thresholds: refer if incurred exceeds USD 250,000, or if incurred exceeds USD 100,000 and the claim is open more than 365 days.

For each claim, think step by step: first state the two threshold checks, then state whether each check is met, then state the overall referral decision.

CLAIM A: Incurred USD 280,000, open 120 days.

CLAIM B: Incurred USD 150,000, open 410 days.

CLAIM C: Incurred USD 95,000, open 500 days.

CLAIM D: Incurred USD 220,000, open 200 days.

Stage 2 output is longer but auditable. The reviewer can scan the per-claim reasoning and confirm the thresholds were applied correctly. Any error is now visible — a threshold misread, an arithmetic slip, a misclassified claim — whereas in Stage 1 the error would be hidden inside a single-word answer.

STAGE 3 — Structured chain-of-thought with explicit output format

For each of the following claims, evaluate referral to large-loss review. Our thresholds: refer if incurred exceeds USD 250,000, or if incurred exceeds USD 100,000 AND open more than 365 days.

For each claim, produce output in this exact format:

CLAIM [letter]: Incurred = [amount]. Days open = [number]. Threshold 1 (incurred > 250,000): [met / not met]. Threshold 2 (incurred > 100,000 AND days > 365): [met / not met]. Referral decision: [refer / do not refer]. Edge case flag: [yes / no — flag if either threshold is within 10% of the boundary].

```
CLAIM A: Incurred USD 280,000, open 120 days.

CLAIM B: Incurred USD 150,000, open 410 days.

CLAIM C: Incurred USD 95,000, open 500 days.

CLAIM D: Incurred USD 220,000, open 200 days.
```

Stage 3 is the version worth using in production review. The structured output can be scanned quickly, the edge case flag surfaces claims near the boundary for closer attention, and the format is consistent across batches — so a reviewer sees the same columns every time and learns where to look for problems.

5.3.4 Self-Consistency

Self-consistency is an extension of chain-of-thought that was proposed to address the reasoning-but-wrong failure mode. The idea is simple: run the same CoT prompt multiple times, then take the majority answer. If the model reaches the same conclusion via several different reasoning paths, confidence in that conclusion rises. If the paths diverge, the actuary has a signal that the task is ambiguous or the model is struggling.

In a web chat interface, self-consistency is manual. The actuary opens a fresh conversation, pastes the same prompt, and records the answer. Repeat three or five times. If all answers agree, proceed. If they differ, the task requires either a more specific prompt or human judgement. The technique is best suited to decisions with a discrete answer — classify a claim as fraud-suspicious or not, select the most appropriate reserving method from a short list, decide whether a policy clause triggers a specific exclusion. It is poorly suited to drafting prose, because there is no meaningful way to take the majority vote across five different paragraphs of narrative.

5.3.5 Tree-Of-Thought

Tree-of-thought prompting extends chain-of-thought further still, asking the model to explore multiple reasoning branches, evaluate each, and commit to the best. It is best suited to planning tasks with clear evaluation criteria — scenario selection for a stress test, decomposition of a reserving analysis into sub-components, identification of candidate pricing segments. It is substantially more demanding of the model than CoT and rarely the right choice when a simpler technique would suffice.

The practical guidance for actuaries is to start with zero-shot, add few-shot when output format or house style matters, add chain-of-thought when the task involves multi-step reasoning that needs to be auditable, use self-consistency only when the stakes of the decision justify the cost, and reach for tree-of-thought only when the task is genuinely a planning problem with branching alternatives.

5.4 Actuarial Prompt Patterns

The techniques in Section 5.3 are general. What follows are four patterns specific to common actuarial tasks. Each is a template that can be adapted to a specific context. Together they cover a substantial share of the tasks where prompt engineering alone — without retrieval or tool use — is sufficient.

5.4.1 The Reserving Commentary Pattern

Task: generate narrative commentary explaining reserve movements, development patterns, or assumption changes. This is the single highest-value prompt engineering application for P&C reserving actuaries, and the one the case study in Section 5.8 addresses in depth. The pattern combines role and context specification with few-shot examples drawn from prior-year filings and a chain-of-thought instruction for any numeric comparisons the commentary must make.

The refinement pattern is the few-shot refinement shown in Section 5.3.2, applied per line of business and per accident year. An actuary running a quarterly reserving cycle can maintain a short library of approved prior-year paragraphs to use as few-shot examples, and swap the input figures to produce the draft commentary for the current cycle.

5.4.2 The Experience Study Summary Pattern

Task: convert a table of experience study results into a written summary suitable for an assumption-setting committee. The pattern uses the model for what it does well — converting structured numbers into fluent prose — while constraining it heavily to the inputs supplied. The constraint matters because the failure mode is confabulation: an LLM asked to summarise a mortality experience study will, if allowed, fill in context about industry norms or recent trends that may or may not apply.

The practical constraint is a closed-world instruction: 'Use only the figures supplied in the input table. Do not reference industry averages, published tables, or trends not present in the data. If a figure is missing or ambiguous, state that rather than estimate.' Closed-world prompts produce less fluent output but more trustworthy output, and for committee papers the second trade is the one that matters.

5.4.3 The Regulatory Response Pattern

Task: draft responses to regulator queries on pricing filings, reserving reviews, or capital model submissions. The failure mode here is the third category of hallucination from Chapter 4 — incorrect interpretation of regulation or actuarial method. Prompt engineering partially mitigates this, but only by forcing the model to ground its response in text the actuary supplies, and by requiring explicit flags where the response requires legal or regulatory judgement beyond what the model can verify.

A reliable regulatory response prompt instructs the model to draft only the technical content, to quote or paraphrase only from the source materials pasted into the chat, and to insert an explicit [LEGAL REVIEW

REQUIRED] flag at every point where the response makes a claim about regulatory interpretation. The flag is a discipline for the reviewing actuary: every instance must be examined before the draft is used. The Chapter 6 RAG architecture is the stronger solution for this class of task, but prompt engineering alone gets much of the way there for narrow, well-scoped queries.

5.4.4 The Peer Review Assistant Pattern

Task: produce a first pass of peer review comments on an actuarial analysis produced by a colleague. The pattern takes a draft analysis pasted into the chat as input, supplies the model with the relevant ASOPs or professional standards as context, and asks for a structured output listing potential concerns, questions for the author, and items that warrant closer review by the human reviewer.

This is the pattern most likely to be resisted by practitioners, and the resistance is legitimate: peer review is a professional responsibility, not a task to be delegated. The pattern's value is as a pre-review — a checklist generator that flags candidate issues for the human reviewer's attention — not as a replacement for the reviewer. Every output must be treated as a hypothesis requiring verification.

5.5 Structured Output

Structured output is the prompting discipline for getting an LLM response into a specified format — a table with fixed columns, a list with fixed fields, a summary with fixed sections. It is distinct from Structured Extraction, the task introduced in Chapter 4 whereby unstructured text is converted into a structured record. Structured Extraction is the end goal; Structured Output is one technique used to achieve it.

In a web chat setting, the two techniques that make structured output reliable are explicit field specification and format demonstration via an example. The prompt below shows both, applied to the task of converting a free-text claim description into a structured record suitable for pasting into a spreadsheet.

STAGE 1 — Weak structured output request

Extract the key fields from this claim description in a structured way:

Insured reports water damage following overnight pipe burst in second-floor server room on 14 November. Initial adjuster estimate USD 340,000 for structural and equipment damage at commercial premises.

Stage 1 produces an extraction — but the field names, the order, and the handling of missing or ambiguous values vary from run to run. Pasting the output into a spreadsheet requires hand-editing every time.

STAGE 2 — Explicit field specification

Extract the following fields from the claim description. Return the result as a two-column table with field name on the left and value on the right.

FIELDS:

- Date of loss (format: DD Month YYYY; use 'not stated' if absent)

- Peril (one of: fire, water, theft, impact, other)

- Estimated loss in USD (integer; use 'not stated' if absent)

- Insured type (one of: commercial, residential, unknown)

- Large loss flag (yes if estimated loss exceeds 250,000; otherwise no; 'unknown' if loss not stated)

CLAIM DESCRIPTION: Insured reports water damage following overnight pipe burst in second-floor server room on 14 November. Initial adjuster estimate USD 340,000 for structural and equipment damage at commercial premises.

Stage 2 produces a consistent two-column table across runs. The fields appear in the specified order, the enumerated values are respected, and the 'not stated' fallback is used when information is missing. The output is close to paste-ready for a spreadsheet, but occasionally the model still takes liberties — inferring the year when only the day and month are supplied, or coding an ambiguous peril description too confidently.

STAGE 3 — Field specification with a worked example and no-guess constraint

Extract the following fields from the claim description. Return the result as a two-column table with field name on the left and value on the right. Use only information explicitly stated in the claim description. Do not infer, calculate, or guess.

FIELDS:

- Date of loss (format: DD Month YYYY; use 'not stated' if absent)

- Peril (one of: fire, water, theft, impact, other)

- Estimated loss in USD (integer; use 'not stated' if absent)

- Insured type (one of: commercial, residential, unknown)

- Large loss flag (yes if estimated loss exceeds 250,000; otherwise no; 'unknown' if loss not stated)

EXAMPLE:

CLAIM DESCRIPTION: Fire damage reported at insured restaurant on 3 March 2025; adjuster estimate USD 120,000 for kitchen equipment replacement.

OUTPUT:

Date of loss | 3 March 2025

Peril | fire

Estimated loss in USD | 120,000

Insured type | commercial

Large loss flag | no

CLAIM DESCRIPTION: Insured reports water damage following overnight pipe burst in second-floor server room on 14 November. Initial adjuster estimate USD 340,000 for structural and equipment damage at commercial premises.

Stage 3 produces a stable output across runs and across similar inputs. When the claim description lacks a year, the date field records '14 November' without inventing a year. When the peril is unambiguous, the field uses the closest match from the enumerated values. The output is paste-ready, and the batch of 200 claim descriptions an actuary might be processing in a single session produces 200 parseable records.

The failure modes are specific. Models can produce nearly-valid output that is broken by a small formatting deviation — an extra blank line, a misplaced pipe character, a field value that includes a pipe character in the text itself. The mitigation is a brief review pass on a sample of the output before relying on the batch, and treating any output that does not paste cleanly as a signal to revise the prompt rather than to hand-fix the output.

5.6 Prompt Chaining

Prompt chaining is the practice of breaking a complex task into a sequence of simpler prompts, where the output of each becomes part of the input to the next. In a web chat setting, this usually means running

the prompts as separate conversations — or separate messages in the same conversation — and copying the output of one step as the input to the next.

The rationale for chaining is direct: LLMs perform more reliably on narrow, well-specified tasks than on broad, compound ones. A prompt that asks for a full SAO narrative in one shot is a compound task; a chain that first summarises the development patterns, then drafts the assumption-change section, then combines the two into a coherent narrative, is three narrow tasks. The chained version is longer to run, but the output is more reliable, and — critically — each step is independently reviewable.

A useful convention in a web chat workflow is to run each step of a chain in a fresh conversation. This prevents the model from carrying forward context from earlier steps that the actuary has not explicitly approved. The discipline is closer to running a sequence of scripts than to a single long chat thread, but it is the discipline that produces reviewable output. The case study in Section 5.8 shows a three-link chain run this way.

Chain design matters. Good chains are linear where the downstream task genuinely requires the upstream output, and are broken into separate threads where the steps are independent. A chain that forces sequential execution of independent tasks is slower without being more accurate. Chains should also include a final integration step that makes the combined output coherent; a chain that simply concatenates its intermediate outputs produces a document that reads like it was assembled from fragments, because it was.

FOR LEADERS: When Prompt Engineering is Not Enough

Prompt engineering is the cheapest and fastest way to get an LLM producing useful actuarial output. It requires no infrastructure, no vector databases, no additional model training. For narrow, well-scoped tasks — drafting narrative commentary, extracting structured records from unstructured text, summarising technical documents for a non-technical audience — it is often the complete solution.

It is not the complete solution when the task requires the model to access information beyond its training data, when the response must be verifiable against a specific source, or when the actuary

needs the model to execute calculations rather than describe them. For those cases, the next three levels of the mitigation hierarchy — Retrieval-Augmented Generation (Chapter 6), fine-tuning (Chapter 7), and tool use (Chapter 10) — are the appropriate investments. The strategic question for a Chief Actuary is not whether to fund prompt engineering but how to sequence the investment across these layers. Prompt engineering first, because it is cheap and reveals which tasks the other layers actually need to solve.

5.7 Common Pitfalls

Five failure patterns recur often enough in actuarial applications of prompt engineering to warrant explicit treatment. Each maps to one or more of the three hallucination categories established in Chapter 4, and each has a corresponding discipline that reduces but does not eliminate it.

The first pitfall is over-reliance. An LLM that produces fluent, confident prose will tend to be trusted more than the underlying reasoning deserves. The reviewing actuary must actively read the output as a draft requiring verification, not as a colleague's considered opinion. The professional standard here is unambiguous: the Appointed Actuary who signs the Statement of Actuarial Opinion bears personal professional liability for its contents. An LLM-generated paragraph in that opinion carries the signing actuary's professional responsibility just as a human-drafted paragraph would.

The second pitfall is confirmation bias. An actuary who expects the model to produce a particular conclusion — because the underlying analysis points that way — will tend to read the output for agreement with that conclusion and miss the places where the model's reasoning has drifted. Model outputs should be checked against the inputs, not against the reviewer's expectations.

The third pitfall is plausible but wrong actuarial reasoning. An LLM that is asked to explain why a reserve should be strengthened will produce a confident explanation even when the data does not support strengthening. The mechanism is not malice or even error in the traditional sense; the model is optimising for fluent continuation of the prompt, and the prompt asked for a strengthening rationale. Prompts that ask for a conclusion produce that conclusion. Prompts that ask for an analysis produce an analysis. The wording matters.

The fourth pitfall is prompt drift. An iteratively developed prompt that started as a focused specification can, after multiple revisions in the same chat window, accumulate contradictions, redundancies, and legacy instructions that no longer apply. Prompts used repeatedly should be saved outside the chat — in a team document, a shared template library, or a prompt catalogue — and rebuilt from first principles when they grow unwieldy. A prompt an actuary cannot scan on a single screen is one that warrants review.

The fifth pitfall is under-specification of failure modes. Good prompts tell the model what to do when things go wrong — what to output when a required input is missing, what to flag when the input contains an inconsistency, when to refuse rather than guess. An LLM without such instructions will default to producing an output of some kind, because that is what it is trained to do. Specifying refusal conditions is as important as specifying success conditions.

GOVERNANCE WARNING: Professional Accountability for LLM-Generated Actuarial Content

Actuarial Standard of Practice (ASOP) No. 56, Modeling, adopted by the Actuarial Standards Board in December 2019 and effective for work performed on or after 1 October 2020, applies to actuaries performing services with respect to designing, developing, selecting, modifying, or using all types of models — explicitly including models developed by others for which the actuary is responsible for the output. An LLM used to draft actuarial commentary, extract structured features from narrative documents, or produce regulatory responses falls within scope. The actuary using the LLM bears the responsibility for understanding the model's intended purpose, its limitations, and the appropriateness of its outputs for the use at hand.

In practice, the reviewing actuary cannot delegate professional judgement to prompt engineering. An LLM-generated paragraph in a Statement of Actuarial Opinion is the signing actuary's paragraph, with the same liability attached. The discipline this chapter recommends — explicit constraints, closed-world prompts, structured output with review, prompt version control, and chain-of-thought reasoning that the reviewer reads — are ASOP 56 compliance tools as much as they are productivity tools.

5.8 From Chat Window to Production

The techniques in this chapter have been presented as they are used in the web chat interface, because that is where most actuarial use of LLMs starts. For any single-user, exploratory, or periodic task — drafting a quarterly commentary, reviewing a single filing, summarising a research paper — the chat window is the right tool. It requires no infrastructure, no engineering support, and no new vendor agreement; the actuary can be productive within minutes of opening the browser.

The chat window stops being the right tool when the task becomes recurring and high-volume — extracting structured records from thousands of underwriting notes, classifying tens of thousands of claims, producing commentary paragraphs for hundreds of portfolio segments every quarter. At that scale, the manual copy-paste-review loop is slower than a short script, and the prompts need to be stored, version-controlled, and invoked programmatically. This is where a provider's LLM API, and equivalent interfaces for other providers, enter the picture.

The same prompt that works in the chat window works through the API with minimal change. The illustrative snippet below shows the Link 1 Keystone Commercial prompt invoked through the Google Gen AI Python Software Development Kit (SDK), as it might be used in a batch script that processes all 48 line-of-business-and-accident-year combinations in a single run. The code is shown for orientation only; Chapter 8 develops the full architectural context in which such code fits.

```
# Illustrative only — full setup covered in Chapter 8
from google import genai

client = genai.Client()   # reads API key from environment

prompt_text = (
    "You are a reserving actuary drafting Schedule P
    ↪  commentary..."
    # (full Link 1 prompt text, as shown in the case study)
)

message = client.messages.create(
    model="gemini-3.1-flash-lite",
    max_tokens=300,
    messages=[{"role": "user", "content": prompt_text}]
)

draft_commentary = message.content[0].text
```

The mental shift from chat to API is smaller than it appears. The prompt is the same text. The model is the same model. The output is the same output. What changes is that the input can be parameterised, the output can be captured programmatically, and the workflow can scale to volumes that a human operator could not process by hand. Chapter 8 takes up the architecture, error handling, and governance implications of that shift; Chapters 9 through 12 build on the same foundation to construct systems in which the LLM is one component among several.

Reader Takeaway

After working through this chapter, the reader can:

- Construct an actuarial prompt with all six components — role, context, instruction, format, constraints, and examples — for a specific drafting or extraction task.

- Apply zero-shot, few-shot, chain-of-thought, self-consistency, and tree-of-thought techniques to the hallucination category each one best mitigates.

- Design a three-link prompt chain that decomposes a compound actuarial task into independently reviewable steps with traceable inputs and outputs.

- Evaluate when a chat-window workflow is sufficient and when the task warrants moving to an API, Retrieval-Augmented Generation, or fine-tuning.

Chapter Summary

Prompt engineering is the first and cheapest layer of the hallucination mitigation hierarchy established in Chapter 4, and for a surprising range of actuarial tasks it is sufficient. The discipline rests on a simple idea: a prompt is a specification, and the more of the task the actuary specifies explicitly, the less the Large Language Model has to guess. Six components — role, context, instruction, format, constraints, examples — structure any good actuarial prompt. Five techniques — zero-shot, few-shot, chain-of-thought, self-consistency, tree-of-thought — match specific failure modes, and each is demonstrated in this chapter through a three-stage refinement that shows a weak prompt becoming a production-quality prompt by addition of specific components. Four patterns — reserving commentary, experience study summaries, regulatory response, peer review assistance — cover a large share of practical use cases. Prompt chaining breaks compound tasks into reviewable steps, usable today in a Claude.ai, Gemini, or ChatGPT chat window with no additional infrastructure. The Keystone Commercial case study demonstrated how a three-link chat-based workflow compresses SAO narrative drafting from 7 working days to 2.5 without degrading the Appointed Actuary's professional review. Chapter 6 builds the next layer of the mitigation hierarchy, Retrieval-Augmented Generation, which addresses

the failure modes that prompting alone cannot reach — tasks requiring the model to ground its outputs in specific source documents the actuary supplies.

Further Reading

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* Advances in Neural Information Processing Systems 35 (NeurIPS 2022), pp. 24824–24837. [The foundational paper for chain-of-thought prompting; essential reading for any actuary using LLMs on multi-step reasoning tasks.]

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). *Language Models are Few-Shot Learners.* Advances in Neural Information Processing Systems 33 (NeurIPS 2020). [The GPT-3 paper that introduced few-shot in-context learning as a general capability; still the clearest exposition of why examples in prompts work.]

Actuarial Standards Board (2019). *Actuarial Standard of Practice No. 56: Modeling.* American Academy of Actuaries. Effective 1 October 2020. [The US ASOP governing actuarial use of models; directly applicable to any LLM-based actuarial workflow where the actuary is responsible for the output.]

Richman, R. (2021). AI in Actuarial Science — A Review of Recent Advances (Part 1 and Part 2). Annals of Actuarial Science, 15(2), pp. 207–258.

Google (2024–). *Prompt design strategies.* Gemini API documentation, ai.google.dev/gemini-api/docs/prompting-strategies. [The working practitioner reference for prompt construction, updated as models evolve; covers both chat-interface and API patterns.]