From Natural Description to Simulation: Large Language Models for Geothermal Engineering

Siying Huang[1], Zhi Hui Celina Kan[1], Michael Gravatt[1,2], Alexander Catalinac[1,2], Cameron Walker[1], Ruanui Nicholson[1,2], Alex de Beer[3], Paul Denny[4], John O'Sullivan[1,2], Oliver J. Maclaren[1,2]

[1] Department of Engineering Science and Biomedical Engineering, The University of Auckland, 70 Symonds Street, Grafton, Auckland 1010, New Zealand
[2] Geothermal Institute, The University of Auckland, 70 Symonds Street, Grafton, Auckland 1010, New Zealand
[3] School of Mathematics and Statistics, University of Sydney, Carslaw Building (F07), New South Wales 2006, Australia
[4] School of Computer Science, The University of Auckland, 70 Symonds Street, Grafton, Auckland 1010, New Zealand

oliver.maclaren@auckland.ac.nz

Keywords: *Artificial Intelligence, Large Language Models, JSON, Validation, Semantic Accuracy, Retrieval Augmented Generation, Chain of Thought, Prompt Engineering, Fine-tuning, Waiwera*

ABSTRACT

Computational modelling in geothermal engineering presents many opportunities for improved automation and validation. For example, manually editing configuration files and meshes for computational modelling is time-consuming and error-prone. In this study, we explore the potential of using large language models (LLMs) to automate typical tasks in geothermal engineering, focusing on generating and modifying structured simulation inputs (in JavaScript Object Notation or JSON format) for Waiwera, a modern geothermal simulator. However, these artificial intelligence (AI) tools come with risks in an engineering context, where precision, reliability and accuracy are crucial. Hence, we consider both syntactic validity and semantic accuracy of LLM generated configuration files. This article first reviews various core concepts of LLM workflows and AI engineering relevant to geothermal engineering. We then present preliminary results of experiments using LLMs for various geothermal modelling tasks, in particular knowledge base retrieval and JSON generation. We consider both open-source and commercial LLMs (e.g., OpenAI GPT-4o, Meta LlaMA 3.3, DeepSeek R1) and discuss progress towards developing a validation framework and implementing AI engineering techniques to improve performance. Initial results indicate improved knowledge retrieval through retrieval-augmented generation (RAG) and consistent generation of structurally valid JSON inputs by LLMs, with minor sensitivity in smaller files. Ultimately, we aim to develop a "virtual geothermal assistant" that improves accessibility and efficiency in geothermal simulation, while documenting best practices for LLM integration in engineering workflows.

1. INTRODUCTION

Prior to the emergence of large language models (LLMs), neural networks (NNs) were already widely applied in various engineering domains, including geothermal engineering (Puppala et al., 2023). For instance, neural networks implemented in an optimisation algorithm demonstrated the ability to identify optimal placement of injection well (Akin et al., 2010). In this context, NNs have been used to extract patterns and detect trends that are often too complex for humans or conventional computational methods to identify. They offer several key advantages, including adaptive learning, self-organisation, real-time processing, and fault tolerance (Han et al., 2021; Torres-Huitzil & Girau, 2017).

LLMs are advanced deep learning architectures pretrained on a large amount of available data (Zhao et al., 2025). At their core, transformers are neural network architectures that use encoder-decoder components and a self-attention mechanism to identify the most relevant parts of a text sequence, enabling them to extract semantic meaning and capture contextual relationships (Vaswani et al., 2017). Transformers engage in self-supervised learning - a process that enables them to infer linguistic patterns, grammatical structures, and factual knowledge from raw data without explicit human-labeled supervision (Kotei & Thirunavukarasu, 2023).

Geothermal reservoir simulation is a critical tool in designing and managing geothermal energy systems (M. O'Sullivan et al., 2001; M. O'Sullivan & J. O'Sullivan, 2025). Modern simulators like Waiwera, an open-source geothermal flow simulator developed at the University of Auckland's Geothermal Institute, allow engineers to model complex subsurface flows and heat transfer, but rely heavily on detailed configuration files (in JavaScript Object Notation, also known as JSON format) to define simulation parameters, physical properties, boundary conditions, initial states, and mesh geometries (Croucher et al., 2020). Manually editing these input files is a complex and error-prone process that demands both domain expertise and careful attention to software-specific syntax, increasing the risk of mistakes such as incorrect unit conversion and missing required fields, which can lead to simulation failures.

LLMs offer a promising avenue for the automation of translation processes to significantly streamline geothermal workflows. LLMs have demonstrated the ability to parse natural language and generate structured outputs, including code and data formats, in various domains (Liang et al., 2025). If an LLM can reliably convert an engineer's description into a syntactically correct and physically meaningful Waiwera JSON input, it has the potential to reduce the effort and expertise needed to set up simulations. However, applying LLMs in engineering contexts poses unique challenges: the output must be correct in structure (to satisfy the JSON schema and simulator requirements) and accurate in content (to reflect the true physical scenario without hallucinated or omitted parameters). Programs incorporating LLMs into their design typically lie on a spectrum from 'workflows' at one end to 'agents' at the other, indicating the degree to which the LLM is responsible for key decisions. As this classification is better viewed as a spectrum rather than a sharp distinction, such programs are often collectively referred to as agentic systems. This motivates our approach of implementing an agentic system for Waiwera, where the LLM is augmented with tools and is capable of autonomously selecting appropriate tools,

generating search queries, and deciding what information to retain in memory.

This paper first presents a review of the key components involved in constructing an agentic LLM workflow, beginning with prompt engineering and progressing to advanced optimisation techniques such as retrieval-augmented generation (RAG) and model fine-tuning, situated within the context of geothermal engineering applications. We then consider preliminary progress towards implementing these ideas in a geothermal engineering context.

2. PROPRIETARY MODELS VS. OPEN-SOURCE MODELS

Proprietary models are LLMs whose weights, architectures, and underlying code are not publicly accessible. These models are known for their strong generalisation capabilities, high accuracy in following complex instructions, and ability to process long contexts. Example models include Google's Gemini series, OpenAI's GPT family, and Anthropic's Claude models (Google, n.d.; OpenAI, n.d.; Anthropic, n.d.-b). However, these models may incur significant costs depending on the API usage and may raise data privacy concerns.

In contrast, open-source models have publicly available weights, enabling local deployment and task-specific fine-tuning (Huyen, 2024). This offers advantages such as greater adaptability, transparency, user control and data protection. However, the process for fine-tuning still requires substantial computational resources - such as high-end GPUs and technical expertise for effective deployment (Alammar & Grootendorst, 2024). Notable examples of open-source models include Meta's LLaMA, Mistral's Mixtral, and Alibaba's Qwen (Meta, n.d.; Mistral AI, n.d.; Alibaba Cloud, n.d.).

Recent benchmark evaluations have shown that some open-source models now match or outperform proprietary models across coding and reasoning tasks. For example, Qwen 2.5-Max has equalled or surpassed fine-tuned proprietary models such as GPT-4o-0806 and Claude-3.5-Sonnet-1022 on benchmarks including Arena-Hard, LiveCodeBench, and GPQA-Diamond (Qwen Team, 2025). These results highlight open-source models as cost-effective, high-performance alternatives for research and production use.

Given these advantages, our work prioritises the use of open-source models where possible and explores their performance across a range of scales (from 1.5B to 70B trainable parameters). Trainable parameters (or weights) are the numerical values within a neural network that are iteratively adjusted during training to minimise the difference between the model's predictions and the true outcomes. They define how input data is transformed as it passes through the network, ultimately shaping the model's ability to recognise patterns, make predictions, and generalise to new data. Model size is particularly important in this context: smaller models are faster and cheaper to run but may struggle with complex reasoning tasks, whereas larger models typically exhibit stronger generalisation, reasoning, and instruction-following abilities, albeit at higher computational cost. This trade-off motivates the exploration of design patterns, workflows, and optimisation techniques that can maximise the utility of both small and large models - a topic addressed in the following sections.

3. AGENTIC SYSTEMS AND COMMON DESIGN PATTERNS

Systems that incorporate LLMs can be positioned along a spectrum, ranging from structured 'workflows' to fully autonomous 'agents', reflecting the degree of decision-making delegated to the model. A workflow follows a prespecified sequence in which LLMs and external tools are coordinated through predefined steps and deterministic code paths (Anthropic, 2024). Typically, this involves retrieving relevant context, then calling the model with that context to generate the desired output. In contrast, LLM-based agents grant the model greater autonomy, enabling it to dynamically select tools, determine action sequences, and manage memory or retrieval strategies to achieve a task. They may pause for human input at checkpoints or under uncertainty, operating in a feedback loop with external tools. This underscores the importance of intuitive toolsets, clear documentation, and operational safeguards such as iteration limits. As this is a spectrum rather than a strict classification, systems are often grouped together as 'agentic systems', typically consisting of an LLM with tool-calling, memory management, and retrieval capabilities that enable adaptive planning and execution with minimal hard-coded intervention (Anthropic, 2024).

We aim to develop an agentic system for the Waiwera geothermal simulator that can: (1) answer queries about geothermal concepts, the Waiwera documentation, and, if applicable, the provided Waiwera model; and (2) modify existing JSON files based on natural language instructions, enabling quick edits like adjusting rock properties or boundary conditions without manual changes.

3.1 Prompt chaining workflow

Prompt chaining is a workflow strategy that decomposes a task into a series of sequential steps, where the output of each step is used as input for the next LLM invocation, as shown in Figure 1 (Anthropic, 2024). This approach improves the accuracy, traceability, and modularity of the final output, although it may increase latency. It is particularly useful for complex interactions with structured files like JSON, where different parts of the file may require independent modifications across multiple LLM calls. In such cases, the initially generated or partially edited JSON file can be passed to a subsequent LLM call for semantic validation or refinement, enabling a more controlled and interpretable editing process. This workflow is also applicable to the JSON modification feature, where a user may request modifications like "change the permeability of all basalt rock types to 1.5 md" or "increase the bottom boundary mass flow by 10%". These commands are decompose into steps such as locating relevant entries, applying transformations, and verifying changes.

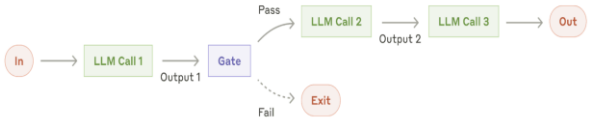

Figure 1: The prompt chaining workflow (from Anthropic, 2024).

3.2 Evaluator-optimiser workflow

Another workflow is the evaluator-optimiser loop, in which an initial LLM call generates a candidate response, and a secondary LLM call evaluates it against predefined criteria, providing feedback or adjustments (Anthropic, 2024). This iterative process is particularly effective in scenarios where well-defined evaluation criteria exist, and iterative refinement is essential for enhancing output quality.

This workflow applies to the JSON validation process, where the evaluator can be an LLM, a tool-based function (such as a JSON schema validator), or a combination of both. Modifications on the JSON file are performed iteratively to ensure that the content is structurally and contextually accurate, as depicted in Figure 2. The validation process involves the LLM Call Evaluator assessing whether the generated JSON is structurally and semantically correct, while the LLM Call Generator refines the output based on the evaluator's feedback.

Figure 2: The evaluator-optimiser workflow (from Anthropic, 2024).

3.3 Tool use

The capabilities of LLMs can be enhanced by integrating tools that enable the agent to perceive its environment and perform actions. According to Huyen (2024), tools can be classified into three categories: tools that perform knowledge augmentation, tools that extend capability, and tools that enable the agent to act on its environment.

Knowledge augmentation tools provide models with access to external resources, helping to construct context and improve responses while reducing hallucinations. Examples of such tools include text retrievers, image retrievers, SQL executors, and web browsing tools. Capability extension tools address the inherent limitations of LLMs by expanding their functionality. These tools might include a calendar, code interpreter, calculator, or translator. Finally, tools that allow agents to act upon their environment enable the agent to perform actions that modify data sources, such as writing or updating information.

In practice, tools are implemented as functions that can be called based on user queries. Function calling generally works by first creating a tool inventory, where each tool is defined by its input and output parameters. By specifying which tools an agent can access, multiple tools can be used per query through an API call.

4. AI ENGINEERING TECHNIQUES

To address the challenge of generating correct simulation inputs, we explore several AI engineering techniques. In this section, we outline the approaches under consideration and relate them to existing literature and examples in similar fields.

4.1 Optimising LLMs

Methods for optimising LLMs for a given task can be broadly divided into two categories: those that utilise the ability of the LLMs to learn purely from text-based instructions (prompting) without updating parameters, and those that involve directly updating the models' weights. The former approach is often referred to as prompt engineering or prompt optimisation, while the latter is known as fine-tuning. Although computationally this represents a key distinction, optimising compound workflows or pipelines involving LLMs often involves both aspects, and recent work has looked at how to carry out both tasks simultaneously (Soylu et al., 2024). Below we discuss these two broad classes of approach in more detail.

4.1.1 Prompt engineering

Prompt engineering is the practice of designing inputs to LLMs to guide them toward producing accurate, relevant, and well-structured outputs. In this section, we discuss three key techniques, system prompting, chain-of-thought prompting, and few-shot learning, which help control model behaviour, improve reasoning quality, and align outputs with task requirements.

System prompting is another important prompt engineering technique. A system prompt provides overarching instructions that define the model's role, behaviour style, and constraints throughout the interaction. Many model providers emphasise that integrating system prompts into an LLM agent can improve performance (Huyen, 2024). For example, Anthropic (n.d.-a) documentation emphasises: "Use system prompts to define Claude's role and personality. This sets a strong foundation for consistent responses." Additionally, they recommend "When setting up the character, provide detailed information about the personality, background, and any specific traits or quirks. This will help the model better emulate and generalize the character's traits." (Anthropic, n.d.-a). For Waiwera simulations, we may integrate system prompting to anchor the model effectively: "You are a geothermal-engineering expert using the Waiwera geothermal simulator. Your job is to answer questions about Waiwera and generate valid simulation-ready JSON files for Waiwera that strictly follow the provided JSON schema." Such role assignment ensures responses are domain-appropriate and consistent.

We can encourage the model to "think" on a prompt for longer, through what is commonly known as Chain-of-Thought (CoT) prompting. CoT means explicitly asking the model to think step by step, guiding it towards a more systematic approach to problem solving. In an experiment conducted by Wei et al. (2022), CoT improved the performance of LaMDA, GPT-3, and PaLM on MAWPS (Math Word Problem Solving), SVAMP (Sequence Variation Analysis, Maps, and Phylogeny), and GSM-8K benchmarks. In the context of assisting JSON generation for Waiwera, rather than asking the model to directly output the final JSON immediately, we can prompt it to break down the task. For example, the first step involves extracting key parameters from the description (domain size, boundary values, sources, etc.), then in a subsequent step, assemble these into the JSON structure.

The final prompt engineering technique is few-shot learning. Few-shot learning refers to teaching a model to learn from

examples in the prompt. Each example provided in the prompt is called a shot. The number of examples provided to the model is limited by the context length; the more examples there are, the longer the prompt will be, increasing the inference cost (Huyen, 2024). In the paper that introduced GPT-3 written by Brown et al. (2020), it was demonstrated that LLMs can learn the desirable behaviour from examples in the prompt, even if this desirable behaviour is different from what the model was originally trained to do. For GPT-3, few-shot learning showed significant improvement compared to zero-shot learning (no examples supplied). It is notable that in some use cases, larger models can perform well even with few examples, due to more robust instruction-following capabilities. As highlighted in a study by Microsoft, few-shot learning only led to limited improvement compared to zero-shot learning on GPT-4 and a few other models (Zhong et al., 2023). In the context of this project, we will aim to balance the number of examples with model size, experimenting across a range of LLMs to identify the most effective trade-off between accuracy and efficiency.

4.1.2 Fine-tuning and custom models

Fine-tuning updates a model's weights by continuing the training of a pretrained base model on a domain-specific dataset (Huyen, 2024). The primary advantage of fine-tuning lies in its potential to enable the model to internalise domain-specific conventions and requirements, thereby improving performance over prompt-based approaches with general-purpose models. For example, an open-source foundation model could be fine-tuned on a curated geothermal dataset to enhance its fluency in Waiwera-compatible JSON syntax and parameter specification.

Despite its potential, fine-tuning presents several practical challenges. It typically requires a substantial number of high-quality training examples to mitigate risks such as overfitting or the hallucination of irrelevant content (Yu et al., 2020). Fine-tuning also demands considerable computational resources and careful hyperparameter tuning to prevent the model from undergoing catastrophic forgetting - a phenomenon where the LLMs forget previously learnt information when training on a specialised task (Luo et al., 2025). Furthermore, there is a risk that a fine-tuned model may become overly specialised, exhibiting strong performance only on tasks closely aligned with its training data while struggling with out-of-distribution inputs (Yu et al., 2020).

A promising approach that we plan to consider in future work is LoRA (Low-Rank Adaptation) on smaller models to reduce resource demands and make experimentation tractable on available hardware. LoRA is a parameter-efficient fine-tuning (PEFT) method introduced by Hu et al. (2021). Prior work by this author has shown that LoRA can match full fine-tuning performance while training only a small fraction of parameters. Instead of updating all the LLM's parameters, LoRA adds small trainable matrices into certain layers of the network. Only these matrices are updated during training, while the original weights remain fixed, lowering memory and computation requirements.

4.2 Knowledge retrieval via retrieval-augmented generation (RAG)

RAG is a technique where the model is supplied with relevant reference materials (retrieved from a knowledge base) alongside the prompt (Hicke et al., 2023). The idea is to give the LLM access to domain-specific information it might not have fully encoded, such as Waiwera's documentation or examples of valid configurations. In this study, we create a knowledge repository that includes the Waiwera documentation and a library of example JSON fragments for various boundary conditions, source terms, etc. RAG often employs methods such as vector databases, which encode documents into numerical representations (called embeddings) that capture semantic meaning. The system can then perform an efficient similarity search to identify the most relevant information, such as sections from the Waiwera documentation that explain how to specify parameter values. These snippets are then appended to the prompt context given to the LLM.

By using RAG, we aim to reduce hallucinations and improve the accuracy of the model's response by providing more relevant context (Ayala & Bechard, 2024). However, it is notable that RAG provides little help in improving the model's deep reasoning abilities; rather, it provides relevant external context to address any knowledge gaps from the LLM's training data, hence, improving the likelihood of producing accurate answers (Liu et al., 2024). The model is less likely to invent non-existent parameter names or physically impossible values if it can refer to the actual documentation. For example, if a user describes a boundary condition, the retrieval fetches the section describing boundary conditions from the Waiwera documentation to be supplied as context.

Literature supports the efficacy of RAG in domain-specific applications. For instance, an LLM was fine-tuned to specialise in building energy modelling using the EnergyPlus simulator, leveraging domain-specific data from RAG to guide its generation process and achieving promising results in producing valid building energy models (Jiang et al., 2024). RAG is also appealing because it avoids retraining the model from scratch with new data; instead, it injects knowledge on-the-fly. This means the system can easily update if Waiwera's specifications change or if we extend to new scenarios, simply by updating the documentation. One challenge with RAG is managing the prompt length: as the amount of retrieved content increases, it can overwhelm the model's limited context window, leading to degraded performance beyond a tipping point (Jin et al., 2025). We will need to carefully select or truncate retrieved information to keep it relevant.

4.3 JSON streaming

Directly prompting an LLM to read, process, or generate complete files via a single prompt often leads to truncation, formatting errors, or overly long outputs that exceed the model's context window (Hosseini et al., 2024; Huyen, 2024). These limitations highlight the need for more controlled approaches.

To address this, an incremental JSON streaming (iJSON) approach can be useful to divide large JSON structures into manageable segments, ensuring syntactic validity at each step (Tobar, 2025). The iJSON method consists of two main components: (1) incremental input streaming, where the LLM processes large JSON inputs in sections, enabling it to fully comprehend all parameters alongside their context; and (2) incremental output generation, which involves generating JSON progressively, validating each chunk iteratively to maintain correctness. A potential extension includes incremental JSON editing, where the model would selectively update a particular section of the existing JSON structure without regenerating the entire file. This iJSON strategy aligns with recommended best practices, ensuring

reliable integration of LLM-generated JSON outputs into complex engineering workflows, while remaining model-independent and straightforward to implement.

4.4 Structured output control

LLMs generate text by predicting the next statistically likely token in a non-deterministic manner (Zhong et al., 2024). Consequently, their outputs can be variable and unpredictable, which poses challenges for seamless integration into automated workflows. To reduce this concern, techniques for structured output control such as controlled generation and schema-driven generation approach can be used by ensuring the outputs follow a predictable pattern.

4.4.1 Controlled generation

Controlled generation constrains model output using templating engines, where prompts are written with placeholders that the model must fill in. Libraries such as Outlines (Willard & Louf, 2023) and LMQL (Beurer-Kellner et al., 2023) guide LLM outputs using templates, which act like forms with "holes" for the model to fill in (Zhong et al., 2024). These templates provide structure and direction, guiding model outputs toward desired properties such as style, tone or structure. However, this guidance is not strictly enforced, meaning the model's response may deviate slightly, as the templates function as more of a structured suggestion rather than a binding rule.

4.4.2 Schema-driven generation

A stronger guarantee can be achieved through schema-driven generation, where outputs are constrained by a formal schema or object specification language (most often JSON) (Zhong et al., 2024). This approach ensures that outputs are not only structured but also parsable and machine-readable.

Libraries such as Outlines (Willard & Louf, 2023) and Instructor (Liu & Contributors, 2024), and frameworks like LangChain (Chase, 2022) and LlamaIndex (Liu, 2022), have abstracted this process for end users. These tools allow users to define an output schema - often using data validation libraries such as Pydantic (Colvin et al., 2025) - and internally translate this schema into natural language prompts that instruct the LLM to return outputs in a parsable format.

5. PRELIMINARY RESULTS

5.1 Model comparison

A selection of open-source and proprietary models were each prompted once under default LLM settings. Their outputs were then manually evaluated for alignment with the Waiwera documentation using three prompts designed to assess both instruction-following ability and contextual accuracy. The first and third prompts focused on simple factual recall from the documentation, while the second tested the models' ability to reproduce provided information:

1. What is the data type for a thermodynamics object in a Waiwera JSON file?
2. Return the first JSON example from the provided context file.
3. What method of simulation mesh does Waiwera use?

The resulting responses were compared against the documentation to determine their correctness and relevance. Table 1 presents a comparative analysis of model performance across the three prompts. A check mark indicates a correct, contextually appropriate response, while a cross denotes an incorrect output, typically due to context limitations or hallucinated information.

The results in Table 1 indicate that smaller models tested significantly underperformed compared to larger models, highlighting the influence of model scale on response quality. This suggests that larger models tend to be more reliable and consistent overall, aligning with prior observations in the field (Brown et al., 2020). Smaller models often require more extensive fine-tuning and task-specific prompting to approach the performance of larger models (Carammia et al., 2024). Therefore, if their outputs are to be made comparable, future work in this project will focus on exploring fine-tuning and prompt optimisation strategies, since smaller models offer a more cost-effective and scalable alternative.

Table 1: Performance of LLMs on basic prompting task

Model Type	Model Name	Size	Prompt		
			1	2	3
Open Source	DeepSeek R1 Distilled Qwen 1.5B	1.5B	×	×	×
Open Source	Gemma Instruct	2B	×	×	×
Open Source	Meta LLaMA 3.2 3B Instruct Turbo	3B	✓	✓	✓
Open Source	Mistral 7B Instruct	7B	✓	✓	✓
Open Source	Qwen 2.5 7B Instruct Turbo	7B	✓	✓	✓
Open Source	Meta LLaMA 3.1 8B Instruct Turbo	8B	✓	✓	✓
Open Source	DeepSeek R1 Distilled Qwen 14B	14B	✓	✓	✓
Open Source	Mistral Small 24B Instruct 25.01	24B	✓	✓	✓
Open Source	Qwen 2.5 Coder 32 Instruct	32B	✓	✓	✓
Open Source	Mixtral 8x7B Instruct v0.1	56B	✓	✓	✓
Open Source	Meta LLaMA 3.3 70B Instruct Turbo	70B	✓	✓	✓
Proprietary	Gemini 2.5 Pro	200B+	✓	✓	✓
Proprietary	GPT 4o	200B+	✓	✓	✓

Legend: ✓ = Correct; × = Incorrect.

5.2 RAG implementation

To implement RAG, we first processed the Waiwera documentation, stored as reStructuredText (.rst) files within the Waiwera Github repository (Croucher, n.d.), for compatibility with automated querying. For this purpose, the document conversion tool Pandoc was installed locally alongside the Python program to facilitate the conversion of the documentation. This setup enabled the automated transformation of .rst files into Markdown (.md) format, ensuring that future updates to the repository documentation could be seamlessly incorporated into the system.

After conversion, the Markdown content was segmented into coherent chunks based on paragraph breaks. These segments were then transformed into vector embeddings and stored in Chroma, an open-source vector database (Chroma Core Contributors, 2025). During inference, the retrieval system searches these embeddings to identify relevant contextual information, which is subsequently provided to the agent to support accurate and informed response generation.

To further enhance the agent's reasoning abilities, we employed DSPy's Chain-of-Thought (CoT) module into the prompting framework (Khattab et al., 2023). This mechanism guided the agent to perform step-by-step reasoning to produce well-structured and context-aware responses. We tested the effectiveness of RAG implemented with DSPy's CoT module by feeding the LLMs with several prompts and comparing the outputs with the Waiwera documentation. The first three prompts are simple quizzes about Waiwera, while the last four prompts require more complex responses, where the LLMs are expected to perform some reasoning before generating their responses:

1. What is the default output fluid fields for water, carbon dioxide and energy ("wce")?
2. What is the data type for a thermodynamics object in a Waiwera JSON file?
3. What is the value type for boundaries JSON object as a Waiwera JSON input?
4. In a two-phase system, why can't pressure and temperature be used as independent primary variables, and what is the role of vapour saturation instead?
5. How do phase changes affect the choice of primary variables, and what challenges might this present in the numerical solution?
6. What is the role of the vector $\boldsymbol{R}(t, \boldsymbol{Y})$ in the time evolution equation, and how does it interact with the mass and energy balances?
7. Why does Waiwera use a finite volume mesh, and how does this relate to the conservation equations?

Comparing model performance without RAG implementation (Table 2) and with RAG implementation (Table 3), we see that when the LLMs are provided with the full context of the Waiwera documentation, the agent consistently generated coherent and logically reasoned outputs.

Table 2: Documentation-based question answering without RAG

Model Type	Model	Size	Prompt						
			1	2	3	4	5	6	7
Open Source	Meta LLaMA 3.2 3B Instruct Turbo	3B	×	×	×	△	△	×	✓
Open Source	Qwen 2.5 7B Instruct Turbo	7B	×	×	×	△	△	×	✓
Open Source	DeepSeek R1 Distilled Qwen 14B	14B	×	×	×	✓	△	△	✓
Open Source	Mistral Small 24B Instruct 25.01	24B	△	×	×	△	△	△	✓
Open Source	Mixtral 8x7B Instruct v0.1	46B	?	△	✓	✓	×	△	✓
Open Source	Meta Llama 3.3 70B Instruct Turbo	70B	×	×	✓	△	△	△	✓
Proprietary	Gemini 2.5 Pro	200B+	×	×	✓	✓	△	?	?
Proprietary	GPT-4o	200B+	×	×	✓	✓	△	×	✓

Legend: ✓ = Correct; × = Incorrect; △ = Partially correct; ? = Requested more context.

Table 3: Documentation-based question answering with RAG

Model Type	Model	Size	Prompt						
			1	2	3	4	5	6	7
Open Source	Meta LLaMA 3.2 3B Instruct Turbo	3B	✓	✓	✓	✓	✓	✓	✓
Open Source	Qwen 2.5 7B Instruct Turbo	7B	✓	✓	✓	✓	✓	✓	✓
Open Source	DeepSeek R1 Distilled Qwen 14B	14B	✓	✓	✓	✓	✓	✓	✓
Open Source	Mistral Small 24B Instruct 25.01	24B	✓	△	✓	✓	✓	✓	✓
Open Source	Mixtral 8x7B Instruct v0.1	46B	✓	✓	✓	✓	✓	✓	✓
Open Source	Meta Llama 3.3 70B Instruct Turbo	70B	✓	✓	✓	✓	✓	✓	✓
Proprietary	Gemini 2.5 Pro	200B+	✓	✓	✓	✓	✓	✓	✓
Proprietary	GPT 4o	200B+	✓	✓	✓	✓	✓	✓	✓

Legend: ✓ = Correct; △ = Partially correct.

5.3 Generating structurally valid JSON

There are two components to validating a Waiwera configuration file: schematically and semantically. This experiment focuses on whether LLMs are capable of producing structurally-valid JSON files without using specialised libraries, regardless of the content of the file. Schema validation matters as it can prevent potential silent errors. These errors can be ignored by Waiwera without a warning, but schema validation will catch it. It also guarantees run-time stability as invalid structure could cause Waiwera to crash or misinterpret data.

For a Waiwera JSON file to be schematically valid, it must conform to the formal Waiwera schema that defines the allowed structure, types, and constraints for every field in the input file. The Waiwera schema (*input_schema.json*), found in the Waiwera Github repository (Croucher, n.d.), is a file that formally defines what a valid Waiwera input JSONs looks like such as types, allowed values, nesting, and constraints. Additionally, all required fields must be present e.g. mesh is mandatory if a mesh object is defined.

Each of the models in Table 4 were given a prompt for a simple geothermal reservoir twice: “Create a JSON file for a geothermal model to be used in Waiwera simulation. The geothermal reservoir has physical dimensions of 16 km by 14 km (horizontal dimensions) by 5 km (depth). The top boundary condition consists of constant pressure of 1 bar and constant temperature of 25 degrees C. The total flow rate of the source input is 100 kg/s with an enthalpy of 1,100 kJ/kg. Use 'placeholder.msh' as the mesh file. Just output the JSON without any comments.”

To ensure structural validity, the generated JSON output was validated against the Waiwera schema. First, the JSON string and the schema definition were loaded. Next, a custom validation function was applied, which passed both objects to the jsonschema library (Berman, 2025). The function then checked compliance with the schema, returning any structural mismatches or formatting errors. As shown in Table 4, some of the smaller models showed signs of variability in structural accuracy. Whereas the larger models performed promisingly. Upon examining the structurally-valid JSON outputs, we noticed that the parameter values from the prompt were all realised and placed in the right place.

Table 4: Generation of structurally valid JSON files by LLMs

Model Type	Model Name	Size	Trial 1	Trial 2
Open Source	Meta LLaMA 3.2 3B Instruct Turbo	3B	×	×
Open Source	Qwen 2.5 7B Instruct Turbo	7B	✓	✓
Open Source	DeepSeek R1 Distill Qwen 14B	14B	✓	×
Open Source	Mistral Small 24B Instruct 25.01	24B	✓	✓
Open Source	Meta LLaMA 3.3 70B Instruct Turbo	70B	✓	✓
Proprietary	Gemini 2.5 Pro	200B+	✓	✓
Proprietary	GPT-4o	200B+	✓	✓

Legend: ✓ = Correct; × = Incorrect.

6. CONCLUSION

This paper presents preliminary progress toward the development of a geothermal agent designed to streamline common modelling tasks, including answering Waiwera and simulation-related queries, translating natural language descriptions into valid JSON configuration files, and editing existing configurations. Key AI engineering techniques, such as RAG, were employed to ground outputs in Waiwera’s documentation, while CoT prompting enhanced the agent’s reasoning and produced more structured, context-aware responses. Looking ahead, the project will focus on curating a dataset of <description, JSON> pairs to enable systematic prompt optimisation and fine-tuning using the DSPy framework. The ultimate objective is to deliver a robust “virtual geothermal assistant” that lowers the barrier to simulation setup, reducing time, effort, and the level of specialised expertise required for geothermal modelling.

ACKNOWLEDGEMENTS

The authors would like to express their sincere gratitude to Adrian Croucher, developer of Waiwera, whose work and contributions laid the foundation for this study.

REFERENCES

Akın, S., Kok, M. V., & Uraz, I. (2010). Optimization of well placement geothermal reservoirs using artificial intelligence. *Computers Geosciences, 36*(6), 776–785. https://doi.org/10.1016/j.cageo.2009.11.006

Alammar, J., & Grootendorst, M. (2024). Hands-On Large Language Models. O'Reilly Media.

Alibaba Cloud. (n.d.). Qwen LLMs. https://www.alibabacloud.com/help/en/model-studio/what-is-qwen-llm

Anthropic. (2024). Building effective agents. https://www.anthropic.com/engineering/building-effective-agents

Anthropic. (n.d.-a). Keep Claude in character with role prompting and prefilling. https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/keep-claude-in-character?

Anthropic. (n.d.-b). Models overview. https://docs.anthropic.com/en/docs/about-claude/models/overview

Ayala, O., & Bechard, P. (2024). Reducing hallucination in structured outputs via retrieval-augmented generation. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, 228–238. https://doi.org/10.18653/v1/2024.naacl-industry.19

Berman, J. (2025). jsonschema 4.25.1. https://pypi.org/project/jsonschema/

Beurer-Kellner, L., Fischer, M., & Vechev, M. (2023). Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages, 7*(PLDI), 1946–1969. https://doi.org/10.1145/3591300

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert- Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language models are few-shot learners. *CoRR, abs/2005.14165*. https://arxiv.org/abs/2005.14165

Carammia, M., Iacus, S. M., & Porro, G. (2024). Rethinking scale: The efficacy of fine-tuned open-source llms in large-scale reproducible social science research. https://arxiv.org/abs/2411.00890

Chase, H. (2022, October). *LangChain*. https://github.com/langchain-ai/langchain

Chroma Core Contributors. (2025). *Chroma: The AI-native open-source embedding database* [Computer software]. GitHub. https://github.com/chroma-core/chroma

Colvin, S., Jolibois, E., Ramezani, H., Garcia Badaracco, A., Dorsey, T., Montague, D., Matveenko, S., Trylesinski, M., Runkle, S., Hewitt, D., Hall, A., & Plot, V. (2025, July). Pydantic Validation (Version v2.12.0a1+dev). https://github.com/pydantic/pydantic

Croucher, A. (n.d.). *Waiwera* [Computer software]. Github. https://github.com/waiwera/waiwera

Croucher, A., O'Sullivan, M., O'Sullivan, J., Yeh, A., Burnell, J., & Kissling, W. (2020). Waiwera: A parallel open-source geothermal flow simulator. *Computers Geosciences, 141,* 104529. https://doi.org/10.1016/j.cageo.2020.104529

Google. (n.d.). Google models. https://cloud.google.com/vertex-ai/generative-ai/docs/models

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *CoRR, abs/2102.04906.* https://arxiv.org/abs/2102.04906

Hicke, Y., Agarwal, A., Ma, Q., & Denny, P. (2023). Ai-ta: Towards an intelligent question-answer teaching assistant using open-source llms. https://arxiv.org/abs/2311.02775

Hosseini, P., Castro, I., Ghinassi, I., & Purver, M. (2024). Efficient solutions for an intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly. https://arxiv.org/abs/2408.01866

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *CoRR, abs/2106.09685.* https://arxiv.org/abs/2106.09685

Huyen, C. (2024). *AI Engineering*. O'Reilly Media.

Jiang, Z., Ma, X., & Chen, W. (2024). Longrag: Enhancing retrieval-augmented generation with long-context llms. https://arxiv.org/abs/2406.15319

Jin, B., Yoon, J., Han, J., & Arik, S. O. (2025). Long-context LLMs meet RAG: Overcoming challenges for long inputs in RAG. *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=oU3tpaR8fm

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2023). Dspy: Compiling declarative language model calls into self-improving pipelines. https://arxiv.org/abs/2310.03714

Kotei, E., & Thirunavukarasu, R. (2023). A systematic review of transformer- based pre-trained language models through self-supervised learning. *Information, 14* (3). https://doi.org/10.3390/info14030187

Liang, H., Kalaleh, M. T., & Mei, Q. (2025). Integrating large language models for automated structural analysis. https://arxiv.org/abs/2504.09754

Liu, J. (2022, November). *LlamaIndex.* https://doi.org/10.5281/zenodo.1234

Liu, J., & Contributors. (2024, March). *Instructor: A library for structured outputs from large language models.* https://github.com/instructor-ai/instructor

Liu, J., Lin, J., & Liu, Y. (2024). How much can rag help the reasoning of llm? https://arxiv.org/abs/2410.02338

Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., & Zhang, Y. (2025). An empirical study of catastrophic forgetting in large language models during continual fine-tuning. https://arxiv.org/abs/2308.08747

Meta. (n.d.). Introducing Meta Llama 3: The most capable openly available LLM to date. Retrieved 2024, from https://ai.meta.com/blog/meta-llama-3/

Mistral AI. (n.d.). Mixtral of experts. Retrieved 2023, from https://mistral.ai/news/mixtral-of-experts

OpenAI. (n.d.). Models. https://platform.openai.com/docs/models

O'Sullivan, M. J., & O'Sullivan, J. P. (2025). Reservoir modeling and simulation for geothermal resource characterization and evaluation. In *Geothermal power generation* (pp. 181-238). Elsevier Science Ltd.

O'Sullivan, M. J., Pruess, K., & Lippmann, M. J. (2001). State of the art of geothermal reservoir simulation. *Geothermics*, *30*(4), 395-429.

Puppala, H., Saikia, P., Kocherlakota, P., & Suriapparao, D. V. (2023). Evalu- ating the applicability of neural network to determine the extractable temperature from a shallow reservoir of puga geothermal field. *Interna-tional Journal of Thermofluids, 17,* 100259. https://doi.org/10.1016/j. ijft.2022.100259

Qwen Team. (2025). Qwen2.5-Max: Exploring the intelligence of large-scale MoE model. https://qwenlm.github.io/blog/qwen2.5-max/

Soylu, D., Potts, C., & Khattab, O. (2024). Fine-tuning and prompt optimization: Two great steps that work better together. https://arxiv.org/abs/2407.10930

Tobar, R. (2025). ijson 3.4.0. https://pypi.org/project/ijson/

Torres-Huitzil, C., & Girau, B. (2017). Fault and error tolerance in neural networks: A review. *IEEE Access, PP,*1–1. https://doi.org/10.1109/ACCESS.2017.2742698

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR, abs/1706.03762*. http://arxiv.org/abs/1706.03762

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E. H., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *CoRR, abs/2201.11903*. https://arxiv.org/abs/2201.11903

Willard, B. T., & Louf, R. (2023). Efficient guided generation for large language models. https://arxiv.org/abs/2307.09702

Yu, Y., Zuo, S., Jiang, H., Ren, W., Zhao, T., & Zhang, C. (2020). Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. *CoRR, abs/2010.07835*. https://arxiv.org/abs/2010.07835

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., . . . Wen, J.-R. (2025). A survey of large language models. https://doi.org/10.48550/arXiv.2303.18223

Zhong, P. Y., He, H., Khattab, O., Potts, C., Zaharia, M., & Miller, H. (2024, January 16). *A guide to large language model abstractions*. Two Sigma. https://www.twosigma.com/articles/a-guide-to-large-language-model-abstractions/

Zhong, W., Cui, R., Guo, Y., Liang, Y., Lu, S., Wang, Y., Saied, A., Chen, W., & Duan, N. (2023). Agieval: A human-centric benchmark for evaluating foundation models. https://arxiv.org/abs/2304.06364