DEPARTMENT OF ENGINEERING MATHEMATICS

An Intelligent Conversational System for Additive Manufacturing Based on Retrieval-Augmented Generation and Large Language Models

Zihao Chen

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Friday 5th September, 2025

Supervisor: Dr. Qunfen Qi

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Zihao Chen, Friday 5th September, 2025

Ethics Statement

Ethics statement: This project fits within the scope of the blanket ethics application, as reviewed by my supervisor Dr. Qunfen Qi.

I have completed the ethics test on Blackboard. My score is 12/12.

Contents

List of Figures

List of Tables

Abstract

Additive Manufacturing (AM) via Fused Deposition Modeling (FDM) using polylactic acid (PLA) and PLA composites has broad applications. However, several challenges remain, such as scattered relevant knowledge and complex parameter settings. While Large Language Models (LLMs) offer potential solutions, direct deployment in such practical domains often results in hallucinations. To address it, this project implemented a chat system based on Retrieval-Augmented Generation (RAG) and LLMs for AM, focusing on FDM using PLA, with basic frontend and backend design.

Approximately 200 research articles on FDM and PLA were collected to construct the RAG knowledge base, supplemented by a manually created HTML table of FDM recommended parameter values for PLA composites from those articles. The system also employed code-driven database management, in which a dedicated expert chat engine was created for each folder of collected PDF files to generate responses solely based on its contents. In addition, 30 open-ended academic queries and 100 parameter recommendation questions were designed to evaluate the system performance.

15 parameter recommendation questions and all 30 open-ended questions were used to examine the system with an LLM-based assessment approach, investigating the output differences of various LLMs (DeepSeek-V3, DeepSeek-R1, GPT-4o, GPT-4.1) under the RAG framework across multiple dimensions (on a scale from 1.0 to 10.0), including Answer Relevance (AR), Context Precision (CP), Completeness, Coherency, and Formatting Quality (FQ). All LLMs achieve high scores in AR ($\geq$ 9.38), with DeepSeek-R1 reaching 9.61. Completeness is highest for GPT-4o and GPT-4.1 (9.59), while CP ($\geq$ 8.97) and coherency ($\geq$ 9.20) also remain strong across all models. FQ is comparatively lower, with scores ranging from 7.04 to 8.31.

Analysis with GPT-4o using defined metrics (Data Coverage Rate, Hit Rate, Precision and Recall) was conducted to investigate the impact of retrieval granularity, controlled by overlap settings, on system performance in 85 FDM parameter recommendation questions with structured tabular data. After optimization, when considering only parameter questions answerable from the tabular data, hit rate, precision and recall all reach 1.0, while the data coverage rate is 0.61, indicating that the structured table contains relevant information for over half of 85 parameter recommendation questions.

Response latency across LLMs was assessed and time consumption at each step during RAG construction on different GPUs was measured. Engineering efforts also included prompt design for both the chat system and the LLM-based evaluator, as well as explicit source attribution incorporated in responses, which mitigated hallucinations and reduced the time spent reviewing the literature.

Additionally, a comprehensive literature review was conducted using a snowballing approach on the development of RAG and LLMs and their deployments in manufacturing. This work offers a valuable reference for future research and practical Artificial Intelligence (AI) applications in the context of data science community.

Supporting Technologies

The main environment of this project was based on `Python 3.9`. I used `LlamaIndex` (an open-source Python library) to build a Retrieval-Augmented Generation (RAG) framework. Specifically, I used:

- `SimpleDirectoryReader()` for caching every document (including literature pages and tabular data), and `Document` for storing content and metadata as a basic unit
- `SentenceSplitter()` for partitioning or parsing documents into chunks
- `VectorStoreIndex()` for converting each chunk into an embedding vector and creating index mappings in the retrieval engine
- `VectorStoreIndex.storage_context.persist()` for storing converted vectors with index mappings
- `StorageContext.from_defaults()` and `load_index_from_storage()` for loading existing vectors without redundant computation

Additionally, I used `OpenAILike()` provided by `LlamaIndex` to call Large Language Models (LLMs), and `HuggingFaceEmbedding()` to access the embedding model.

I also used many open-source `Python` libraries. Specifically, I used:

- `torch` to enable accelerated computation with MPS or GPU
- `random` to set random seeds
- `time` to record the execution time of various steps in building the RAG framework and latency of LLMs
- `logging` to efficiently record system runtime states
- `Path` to access file information such as name and extension
- `typing` to standardize function inputs and outputs for best practices
- `json` to store the conversation history between LLMs and users
- `argparse` to simulate command line arguments in the code
- `FastAPI` to construct the backend service

The frontend was developed using following open-source languages and libraries:

- `HTML` language for building frontend components
- Basic `CSS` language for component layout and colors
- JavaScript (Vue 3) language for enabling dynamic user interaction on the frontend
- `axios` library in the frontend for sending HTTP requests to the backend
- `Font Awesome 6` for adding publicly available icons on the frontend
- `marked` for rendering markdown tags from LLM responses in the frontend

The local environment was an Apple M1 Pro 14-inch machine (16GB, macOS 14.7.6) with the Metal Performance Shaders (MPS). GPU experiments were run on Featurize with following configurations:

- RTX 4090, 14 x AMD EPYC 7453, 23GB
- RTX A6000, 30 x AMD EPYC 7742, 48GB
- RTX 3090, 6 x Xeon Gold 6142, 24GB
- Tesla V100, 6 x Xeon Gold 6142, 16GB

Chapter 1

Introduction

1.1 Context and Motivation

Additive Manufacturing (AM), also called 3D Printing (3DP), is a production method of arranging layers of substances to generate tangible parts based on predefined computational designs [24]. As a core technology of the Fourth Industrial Revolution (Industry 4.0), it can promote efficient and progressive manufacturing, from prototyping to product realisation [49]. In contrast to classic subtractive manufacturing [24], AM is capable of providing complex and personalized components to stakeholders while maintaining cost-effectiveness [30, 58], thereby being widely utilized in engineering disciplines such as aerospace and biomedical solutions [29, 24, 49, 25].

Among various AM technologies, Fused Deposition Modeling (FDM) is broadly employed and highly valued [24, 43]. It forms items by depositing thermoplastic filaments like polylactic acid (PLA), one layer at a time via the heated nozzle. This paradigm is considered advantageous because, in addition to general benefits previously mentioned [49], it offers opportunities for non-expert users to create components [39] under safe conditions [25]. However, FDM faces a range of challenges. At a general level, the fabrication process itself can be quite involved, requiring careful attention to multiple steps [29]. At a more specific level [58], it involves a variety of operational parameters that may have a substantial impact on the reliability of fabricated objects [5, 31]. Moreover, current parameter recommendation processes are complicated, and related information is fragmented, posing a specific threshold for operators [25, 30, 13, 58]. Notably, as PLA is increasingly combined with various materials for broader applications, parameter settings and associated knowledge requirements have become more complex [24], further aggravating these challenges [49, 39, 29], for both researchers and non-expert users.

With rapid advancements in Large Language Models (LLMs) recently, their strong abilities in knowledge inference and conversational interaction [6, 59, 43, 58, 15, 8, 22, 54, 27] have offered a promising solution to the challenges mentioned above [31, 13, 35, 25]. However, directly deploying LLMs may not be the optimal choice. For one thing, their universal and static training datasets may prevent them from providing granular guidance when addressing questions in particular fields [63, 5]. For another, LLMs may return outputs that seem reasonable but are actually incorrect, known as "hallucinations" [35, 8, 22]. Introducing Retrieval-Augmented Generation (RAG) to LLMs could substantially mitigate these issues, where the LLM is given retrieved literature as reference to inform its answers [5, 6, 54, 43, 31, 59, 15, 14, 44, 58, 28, 63, 30, 35].

The RAG module also brings additional advantages. First, it makes LLM responses, especially numerical parameter recommendations, explicitly linked to specific literature sources [15, 6, 14, 54]. Second, both users with and without relevant expertise can efficiently gain the reliable and authoritative information they need [58] in Natural Language (NL) [35], whether it is comprehensive operational guidance or certain FDM parameter suggestions [43]. Finally, it enables scholars to quickly conduct literature reviews and dynamically incorporate new publications [49, 39, 15, 14, 58, 6, 5, 31, 28, 30]. These features reflect the trend of combining Artificial Intelligence (AI) and AM, as promoted by Industry 4.0 and even Industry 5.0 [25, 35, 53, 51].

1.2 Project Aims and Objectives

The aim of this project is to develop an intelligent conversational system for AM based on RAG and LLMs, with a particular focus on FDM using PLA, supporting academic queries and FDM parameter

recommendations.

The project objectives are as follows:

- Collect and organize relevant literature on FDM and PLA to build RAG retrieval resources, managed in separate folders by distinct indexing engines.
- Design open-ended academic questions for FDM and PLA domains, and FDM parameter recommendation questions for PLA and PLA–filler composites.
- Establish assessment methods for different task types, utilizing an LLM-based evaluation (using DeepSeek-V3 as the evaluator) to compare the performance of four LLMs (DeepSeek-V3, DeepSeek-R1, GPT-4o, and GPT-4.1) on open-ended academic queries across multiple dimensions (Answer Relevance, Context Precision, Completeness, Coherency, and Formatting Quality), and a metric-based evaluation (Data Coverage Rate, Hit Rate, Precision, and Recall) for parameter recommendation tasks.
- Evaluate and compare the latency of different LLMs within the RAG system.
- Investigate acceleration effects of different GPUs on RAG engine construction, benchmarked against the local environment.
- Explore the impact of different retrieval granularities, such as varying overlap sizes, on parameter recommendation tasks.
- Implement the integration between frontend, backend, and core conversational system functions to deliver a complete data science product.

1.3 Report Structure

The remaining report is organized as follows:

- Chapter 2 offers a comprehensive literature review and identifies intended project contributions.
- Chapter 3 presents the overall methodology of this project.
- Chapter 4 provides a critical analysis of experimental results, quantitatively and qualitatively.
- Chapter 5 evaluates the system with different GPUs, chunking strategies and related work. Besides, validity threats are discussed and key research findings are critically summarized.
- Chapter 6 concludes main achievements and contributions, and shares some possible suggestions for future work.

Chapter 2

Background and Related Work

This chapter presents a literature review of this project. Initially, fundamental concepts in Natural Language Processing (NLP) and AI are introduced in Section 2.1. Then, typical Pretrained Language Models (PLMs) and LLMs are thoroughly presented in Section 2.2 and Section 2.3 respectively. The framework of RAG is described in Section 2.4. Moreover, Section 2.5 summarizes related works and Section 2.6 critically discusses limitations of existing studies and intended contributions of this project.

The literature review process was as follows: First, keywords were searched on Google Scholar to examine RAG-LLM studies in AM, research on hallucinations, and review papers on RAG and LLMs. Then, four papers were selected as starting points for backward snowballing, as listed in Table 2.1. The backward snowballing process starting from AMGPT [5] was visualized in Appendix A.1. Forward snowballing was also conducted for AMGPT to identify more engineering-related RAG-LLM studies. In addition, several LLM technical reports were included.

Table 2.1: Starting Set Articles for Backward Snowballing

Article Title	Reference
AMGPT: a Large Language Model for Contextual Querying in Additive Manufacturing	[5]
A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models	[54]
A survey of GPT-3 family large language models including ChatGPT and GPT-4	[27]
Retrieval-Augmented Generation for Large Language Models: A Survey	[15]

2.1 Foundations of General Theories and Techniques

2.1.1 Natural Language Processing and Word Tokenization

NLP, also called "Computational Linguistics", consists of numerous mechanisms and procedures, enabling computers to emulate the way people handle textual information [43]. One major use of NLP is Question Answering (QA), which involves retrieving appropriate content from textual resources and systematically generating answers according to inquiries [41]. In most cases, linguistic data typically undergoes tokenization, a phase that segments continuous content into more granular analytical pieces, termed "tokens" [43].

2.1.2 Early Rule-Based Solutions for Question Answering

Initial NLP applications like QA [41] handled domain-specific texts primarily through a set of rules crafted by experts [27]. However, this human-engineered approach was both labor-intensive and costly, and maintaining or updating the entire tool proved to be quite challenging. Moreover, such systems were only capable of addressing requests that conformed to pre-established templates, with limited adaptability. Sometimes, QA products merely pointed to associated literature, without offering straightforward responses or detailed explanations [58, 2, 63].

2.1.3 Word Embeddings

Word embeddings, a specific form of word encoding, enable the mapping of **tokens** to mathematical expressions [5, 6, 59, 28], thus illustrating semantic associations [55, 43]. Each token is assigned a numerical vector, where each dimension corresponds to a quantitative feature [40], which is established on raw

training datasets [47]. In most cases, embeddings are constructed on the "Distributional Hypothesis" that words occurring in analogous linguistic environments are more likely to exhibit analogous interpretations [55, 43]. Therefore, tokens related in meaning or structure tend to hold vectors in closer proximity [47]. There are three main types of embeddings: "traditional", "static" and "contextual" [40]. The first two are described in Appendix A.2.

In contrast to static embeddings, **contextual embeddings** can adjust the token vector according to its surrounding terms [40]. One major advantage is the embedding of a polysemous word can be modified by context, rather than having a constant representation as in static embeddings. There are two key methods to produce contextual embeddings: auto-encoding and auto-regressive. BERT [11] is a representative auto-encoding model [5], whereas GPT-1 [45] exemplifies the auto-regressive approach [55, 40]. These models are discussed later in Section 2.2.

2.1.4 Deep Learning and Reinforcement Learning

The advancement of Graphics Processing Units (GPUs) has greatly enhanced the efficiency of parallel computation, making "Deep Learning" (DL) possible [41]. DL is a Machine Learning (ML) method based on Artificial Neural Networks (ANNs) [51], and can involve a substantial number of learnable parameters. Progress in DL has promoted the study of NLP and embedding methods [41, 40, 55, 27]. Those parameters are optimized via "Gradient Descent" using "Backpropagation" [48]. The gradient, representing the rate of change of a function, quantifies the extent to which a modification in one parameter impacts the others. One representative DL model is Multi-Layer Perceptron (MLP), also called Feed-Forward Network (FFN) [56], which is composed of multiple interconnected modules, such as input, hidden and output layers, discovering complex relationships within datasets [51, 27].

Recurrent Neural Network (RNN) is an architecture that utilizes embeddings as features to handle [56] text sequentially, one token at a step [41]. Specifically, by synchronizing token positions with time steps, it produces a sequence of hidden states h_t, each of which depends on both the previous hidden state h_{t-1} and the input at position t [56]. Based on this recurrent paradigm, the encoder-decoder architecture was introduced, where the encoder maps the input to a specific vector and the decoder constructs responses from that vector [27]. Then the attention algorithm was incorporated into encoder-decoder architectures, which makes the model focus on important input parts and generate satisfying content [41, 43]. However, the lack of parallelism in recurrent mechanisms and gradient issues during training on long sequences have always constrained their efficiency. Transformer architecture addresses these limitations [56, 47, 6, 27, 45, 5].

Reinforcement Learning (RL) is also a subset of ML, allowing models to consider favorable actions within a given situation, training them based on rewards or penalties [51].

2.1.5 Transformer Architecture

Transformer abandons recurrent structures and is fundamentally based on the self-attention mechanism, enabling efficient parallel computation and robust processing of longer sequences [47, 41, 27, 56].

As shown in Figure 2.1, the left part is the encoder and the right part is the decoder. The input sequence $(x_1, \dots x_n)$ is first tokenized using Byte-Pair Encoding (BPE) and is added positional information using Positional Encoding [47, 27]. Then it is projected into an intermediate vector $z = (z_1, \dots z_n)$ [6]. The decoder returns the output sequence $(y_1, ...y_m)$ token by token in an **auto-regressive** manner. Each subsequent token is a function of previously generated tokens [56]. Specifically, training data is fed into the encoder, where it passes through the Multi-Head Attention (MHA) layer and FFN, each equipped with residual connection and layer normalisation [27]. The decoder adopts a similar architecture [45], but applies an extra **masked** MHA to the target sequence during training and evaluation. This ensures each output token can only attend to previously generated tokens, forming the basis of the auto-regressive framework, also known as "causal self-attention". The final output scores are fed into the softmax layer to produce **probabilities**, from which the most appropriate next token is selected for generation [47, 27, 63, 28]. Dropout is also applied at connections between modules. The entire process is repeated through multiple stacked encoder-decoder frameworks [41, 9, 5], which progressively refines token representations before producing the final output sequence [56, 27, 4].

The MHA layer concatenates multiple self-attention results, allowing for a richer representation of tokens. Residual connections add the input of the previous layer to its output, mitigating performance degradation [20, 47, 27]. Dropout randomly deactivates neurons, ensuring most units are adequately trained, thereby reducing overfitting [41]. In three-dimensional data $(sequence, batch, feature)$, layer

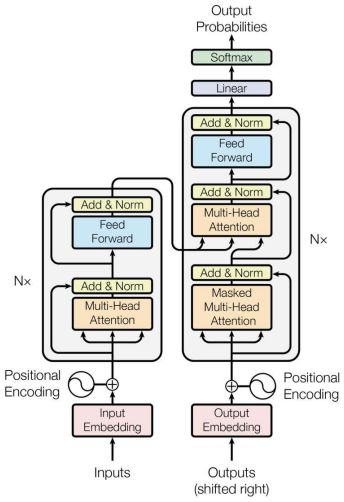


Figure 2.1: Transformer Architecture [56]

normalisation computes the mean and variance along the feature dimension for each sequence, and normalizes features accordingly. This helps stabilize training and accelerates model convergence [3, 56, 27]. The softmax equation is listed in Equation (2.1). The softmax value for the j-th output dimension is calculated by dividing the exponentiated weight for that dimension by the sum of the exponentiated weights across all N output dimensions [34].

$$\text{softmax}(z_j) = \frac{\exp(z_j)}{\sum_{k=1}^{N} \exp(z_k)} [34] \tag{2.1}$$

The self-attention equation is listed in Equation (2.2). The embedding vector of tokens is multiplied by three separate weight matrices w_q, w_k, w_v, to generate corresponding Query (Q), Key (K), and Value (V) matrices. Subsequently, Q and K, having the same dimension d_k, undergo a dot product operation, and the result is scaled by the inverse square root of d_k to ensure stable gradients in training. This scaled output is then converted into a probability distribution by a softmax function. Finally, this distribution is used to weight the V, producing final attention scores. In MHA of encoders, Q, K and V are all derived from the previous encoder output. In masked MHA of decoders, Q, K and V are also from the previous decoder output and the masking operation is applied before softmax to ensure the auto-regressive. In general MHA modules of decoders, Q is from the previous masked MHA output, while K and V are from the final encoder output [27], connecting the encoder and decoder [56].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V [56] \tag{2.2}$$

2.1.6 Transfer Learning

Human data labelling for NLP tasks is resource-intensive, especially since DL models typically need large-scale labelled datasets [51]. Transfer Learning (TL) has emerged as an acceptable strategy for this dilemma. TL refers to the process of applying representations acquired from one task to another task in the same general field, reducing the need to initialize networks without any prior knowledge[27, 45, 47, 50]. Building on this and benefiting from the advanced Transformer architecture [4], three main types of PLMs are developed in NLP: encoder-only, decoder-only, and encoder-decoder models [27, 6, 43, 5].

2.2 Typical Pretrained Language Models

2.2.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) employs only Transformer encoders to simultaneously capture bidirectional sequence dependencies [40, 43] as an auto-encoding approach. It learns universal word knowledge through two pretraining objectives: Masked Language Modelling (MLM)

and Next Sentence Prediction (NSP) [11]. The final encoder output is regarded as contextual embeddings [55].

BERT is pretrained from unlabelled datasets, where labels are automatically formulated according to pretraining objectives without manual annotation [27, 11, 45]. It is particularly suitable for utilizing large amounts of unlabelled internet corpus [46], and this pipeline is now referred to as Self-Supervised Learning (SSL) [47, 27, 55, 7]. BERT only requires adding output modules and updating some pretrained parameters with available task-related samples to perform new downstream Natural Language Understanding (NLU) applications such as text classification and relation extraction [6, 5]. This kind of process is also known as fine-tuning [11, 27, 55, 45, 4, 7, 44].

2.2.2 GPT-1 and GPT-2

Generative Pre-Training of a Language Model (GPT-1) employs only Transformer decoders to generate the next token based on previous tokens, also termed "language modelling objective" during pretraining [47, 7, 43]. Its equation is listed in Equation (2.3). $\mathcal{U} = \{u_1, \ldots, u_n\}$ refers to the token sequence in unlabelled data. k specifies the number of previous tokens used as context for prediction and P indicates the conditional probability, parameterized by neural network weights Θ. These weights are iteratively refined via backpropagation to maximize the likelihood objective $L_1(U)$ [45, 55].

$$L_1(\mathcal{U}) = \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta)[45] \tag{2.3}$$

This unidirectional language modelling approach, known as auto-regressive [55], could be considered more challenging than bidirectional language modelling methods like BERT, since the model only has access to information from one direction for prediction. Nevertheless, this constraint may encourage the model to learn more promising word representations [45, 11]. After fine-tuning, GPT-1 can be applied to both Natural Language Generation (NLG) and NLU applications [6, 5]. However, its fine-tuning process [27] is relatively complex, since it needs to modify the input format with special tokens to help the model recognize input boundaries in downstream tasks, as GPT-1 is pretrained on continuous document streams. GPT-2 addresses this issue by not only utilizing larger and higher-quality pretraining datasets with refined Transformer decoders, but also enabling **problem descriptions, inputs and outputs** to be encoded as part of a continuous sequence like $p(\text{output}|\text{input}, \text{task})$, without additional modifications to the input format [45, 46].

2.2.3 T5

Text-to-Text Transfer Transformer (T5) is an encoder-decoder based PLM. Based on the $p(\text{output} \mid \text{input}, \text{task})$ template of GPT-2, T5 refines an integrated paradigm in which both the input and output are represented as NL, enabling both pre-training and fine-tuning to be conducted within the same framework. This unified format also lays a solid foundation for further advancements of LLMs [7]. T5 uses the maximum likelihood objective as listed in Equation (2.3) and cross-entropy loss as listed in Equation (2.4) for training [47]. The index i indicates the i-th token in the vocabulary corresponding to the possible next token, where p denotes the estimated probability and y represents the true probability. In most cases, the ground truth y is 1 for the correct token and 0 otherwise [60].

$$CE(y, p) = -\sum_{i=1}^{N} y_i \log(p_i)[60] \tag{2.4}$$

2.3 Typical Large Language Models

2.3.1 GPT-3 and InstructGPT

One drawback of this fine-tuning paradigm is its reliance on obtaining a certain amount of annotated data for downstream applications, constraining practical deployments of PLMs. Studies have found the scale of pre-training data and architecture both contribute to capabilities of PLMs [27, 50, 11, 47, 62, 61, 7, 43, 25, 9, 37, 33]. Based on this, GPT-3 was introduced, performing tasks solely through NL queries without gradient adjustment or fine-tuning, by leveraging Few-Shot (FS) learning [55, 42, 63]. This approach enables GPT-3 to demonstrate the In-Context Learning (ICL) ability [27, 4].

During pre-training, ICL enables models to gain recognition capabilities for task requirements, allowing them to identify user instructions expressed in NL optionally accompanied by some task examples, and to complete various objectives by generating tokens at inference time [62, 25, 33]. FS means the model receives input prompts along with several task examples as context. Zero-Shot (ZS) indicates the model is only provided with input prompts without any example demonstration [4]. Since GPT-3, later large-scale language models are typically termed LLMs, to differentiate them from smaller PLMs such as BERT and GPT-2 [27, 58, 5, 63].

InstructGPT utilizes Reinforcement Learning from Human Feedback (RLHF) to guide GPT-3 in generating more fluent and human-like text, improving alignment with user instructions [43, 54]. This mitigates the limitation of GPT-3 pre-training, which only optimizes for next-token prediction without explicitly considering adherence to task instructions in downstream scenarios [42, 27, 4, 5]. RLHF comprises three main steps. **First**, Supervised Fine-Tuning (SFT) is conducted on the LLM using a set of human-written examples that reflect ideal QA pairs and incorporate required prompts [7, 15]. The resulting LLM after this training is referred to as the **reference model** [52, 42]. **Next**, this reference model is used to generate multiple answers for a given question, which are then rated by human annotators. These annotated data are subsequently utilized to train a **reward model** that can decide which output better aligns with annotator preferences [7]. **Finally**, RL with Proximal Policy Optimization (PPO) is applied to further fine-tune a new copy of the reference model using the reward model, resulting in what is known as the **policy model**, so that its outputs better align with reward model preferences [52]. This policy model is the final required model. A 1.3B-parameter InstructGPT trained with RLHF can produce more satisfactory responses than the original 175B-parameter GPT-3 [42].

In the PPO algorithm, the policy model is initialized with parameters of the reference model. For each training question, both policy and reference models generate a single corresponding answer. These QA pairs are fed into the reward model, which assigns scores to their outputs. The difference in their scores is then passed to the **Generalized Advantage Estimation** (GAE) and **Advantage** (A) modules to update policy model parameters through backpropagation, encouraging the policy model to produce responses with higher scores. To prevent "reward hacking" [38], where the policy model might simply cater to the reward model for higher scores at the expense of output quality, a KL divergence penalty is introduced between the policy model and reference model, regulating their token distributions to remain similar. In addition, a **value model** is employed to ensure stability during training [52].

2.3.2 GPT 4o and GPT 4.1

Based on RLHF, GPT-4 further expands the training data and model scale, and is capable of accepting both image and text inputs to generate textual responses. However, for GPT-4 and some subsequent GPT products, OpenAI has not released any technical detail [1, 7]. GPT-4.1[1], an iteration of GPT-4, offers the same input–output pipeline as GPT-4, but likewise lacks publicly available technical specifications. GPT-4o implements end-to-end training on text, audio and image data, and is capable of processing multimodal inputs and generating outputs accordingly. However, the technical report for GPT-4o only shares the use of higher-quality data [23]. Such LLMs are categorized as closed-source, accessible only via APIs and at a cost without technical details [27]. Additionally, reports for these models consistently highlight considerations regarding ethical issues [1, 23, 42, 4, 33].

2.3.3 DeepSeek V3

DeepSeek-V3 is a LLM based on Transformer decoders, featuring the enhanced Mixture-of-Experts (MoE) architecture and the Multi-Head Latent Attention (MLA) mechanism, with the Multi-Token Prediction (MTP) training objective [38]. The traditional MoE approach replaces FFN layers with MoE layers, each comprising numerous expert networks. These experts are assumed to specialize in different knowledge domains. Structurally, an expert network is equivalent to an FFN layer, and each token is routed to one or two experts for output [9].

DeepSeek-V3 introduces DeepSeek MoE: a more fine-grained allocation of parameters to a greater number of smaller expert networks, under the same parameter amount and computation budget [4]. This enables more precise modelling of distinct knowledge areas. Additionally, some experts are designated as shared experts that are consistently activated, primarily responsible for capturing and integrating general knowledge across contexts, thereby reducing redundancy among the other experts. Furthermore, the expert routing strategy is refined to ensure sufficient training for each expert, while also improving

[1] https://openai.com/index/gpt-4-1/

the utilization of computational resources [37]. DeepSeek MoE alleviates the burden of floating-point computations during inference by utilizing both smaller shared and routed experts [9, 38, 4].

When performing inference with a decoder using MHA, the relationship between the current token and all previous tokens is computed at each step [56]. In practice, K and V embeddings of preceding tokens can be cached, so that when predicting subsequent tokens, these cached embeddings can be directly accessed without recomputation. This approach significantly improves inference speed, but leads to considerable GPU memory consumption. The MLA mechanism of DeepSeek-V3 reduces cache overhead and further improves inference performance by applying "low-rank key-value joint compression" to cached K and V embeddings, thereby reducing the dimensionality of the KV cache [37, 38].

As illustrated in Equations (2.5), the hidden state vector h_t of the current token (where t denotes the token index in the sequence) is multiplied by the **D**own-projection matrix of the **K** and **V** cached embedding W^{DKV}, to obtain one compressed latent vector for both K and V, c_t^{KV}. This latent vector is then projected separately by the up-projection matrices for K and V, W^{UK} and W^{UV}, to generate vectors k_t^C and v_t^C used during inference. In this process, only c_t^{KV} needs to be cached. Also, the Q vector undergoes the similar procedure to reduce computation resources during training: it is first multiplied by the down-projection matrix W^{DQ} to obtain a compressed latent vector of Q, c_t^Q, which is then projected by the up-projection matrix W^{UQ} to produce q_t^C. Besides, coupled Rotary Position Embedding (RoPE) is employed to incorporate positional information into the embedding [37, 38].

$$\mathbf{c}_t^{KV} = W^{DKV}\mathbf{h}_t, \quad \mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV}, \quad \mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV}, \quad \mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t, \quad \mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q [37] \tag{2.5}$$

For an input sequence, MTP employs D sequential modules to predict D target tokens. Specifically, the i-th module takes as input the segment starting from the i-th token in the sequence. For example, the first module receives (t_1, t_2, t_3, t_4) as input and predicts (t_2, t_3, t_4, t_5), while the second module takes (t_2, t_3, t_4, t_5) as input, and so on. Main parameters of the "first module" are then passed to the subsequent module, effectively transferring its understanding of **the input segment starting from the i-th token**, and providing a foundation for the next module to predict subsequent tokens. In this way, MTP not only improves training efficiency by allowing the model to learn from multiple labels, but also enables the generation of multiple tokens at once during inference and accelerates decoding [38].

In addition, the Group Relative Policy Optimization (GRPO) algorithm is used for RLHF, instead of the PPO used in InstructGPT. In this case, for each question, the reference model generates one answer, while the policy model produces multiple answers for evaluation by the reward model. Score differences between two models are normalised by "subtracting the group average and dividing by the group standard deviation", and then passed to the A modules for parameter updates via backpropagation. This normalisation replaces the value model used in PPO, reducing memory and computation overhead, since the value model is usually the same size as the policy model [37]. The position of the KL divergence module has also been optimized [52]. Besides, DeepSeek-V3 incorporates reasoning capabilities of DeepSeek-R1 through knowledge distillation as post-training [38, 21, 50, 4].

2.3.4 DeepSeek R1 Zero and DeepSeek R1

DeepSeek-R1 Zero employs the GRPO algorithm on DeepSeek-V3 **without** SFT to train the model for reasoning tasks, using the rule-based reward model that encourages the separation of thinking words and answer words in its outputs through special tokens such as <think>and <answer>[17]. The training of DeepSeek-R1 begins by establishing a *model* dedicated to generating reasoning data. Initially, a limited amount of high-quality reasoning data, obtained through manual review, is used to perform SFT on DeepSeek-V3 as the "cold start" [54]. Subsequently, following an approach similar to R1 Zero, this cold-started model is further trained, and its "checkpoints" are used to generate enough additional **reasoning data**. In addition, Chain-of-Thought (CoT) is applied to DeepSeek-V3 to generate high-quality **non-reasoning** data. These two main types of data are then used to perform further SFT on the standard DeepSeek-V3. Finally, the reward model and GRPO are applied to train that V3 as the final R1 model. One advantage of DeepSeek-R1 is its ability to generate high-quality thinking process words as explicit reasoning data [17].

2.3.5 Chain of Thought and Prompt Engineering

When CoT was first proposed, it meant providing a few samples containing thinking process words as part of the input in the ICL setting, so that LLMs could derive similar reasoning steps [33, 35]. CoT divides

a task into successive stages without updating parameters, offering an interpretable way to observe and understand how it produces a specific answer [62, 44, 54, 63]. This paradigm overcomes the dilemma where increasing the model scale does not enhance reasoning ability [33]. [33] also demonstrates that by carefully designing CoT prompts, for example, by adding "Let's think step by step" before each response, LLMs are able to exhibit ZS learning abilities in reasoning tasks, without the need for additional samples including explicit reasoning processes. This is known as zero-shot prompting and is effective across various domains, not just reasoning [7, 27, 58, 5]. Building on this, the concept of prompt engineering was introduced [33], which involves designing detailed prompts, such as explicitly defining objectives, for LLMs to improve output quality [44, 58, 43, 25, 54, 5].

2.3.6 Limitations of Large Language Models in Domain-Specific Tasks

When applying LLMs to domain-specific NLP tasks, several critical issues arise [15, 22, 27]:

- The key obstacles to utilizing LLMs in high-risk scenarios are hallucinations and the lack of source attribution [1, 36, 15, 8, 28, 43, 27], which stem from the auto-regressive nature of their training paradigm [54, 44].
- LLMs are generally trained once with a fixed dataset and can only be updated later via SFT, making it difficult to rapidly update knowledge at scale [57, 22, 14, 54, 15, 8, 43].
- For most organizations, training such models from scratch or deploying them locally is impractical due to substantial computational and data requirements [22, 14, 54, 58, 43], even for SFT [15, 5]. Moreover, as many LLMs [1, 23] are closed-source, inference is only possible via API [26].

2.4 RAG

RAG offers a practical solution to these issues. In the context of LLMs, it denotes the process of integrating authentically human-authored materials retrieved from external sources into prompts used for ICL, enabling the model to leverage this knowledge to produce content that meets specific requirements [8, 14, 36, 58, 63, 5, 6, 59]. Conceptually, this framework can be divided into three principal stages: Indexing, Retrieval, and Generation [15, 53, 22].

2.4.1 Process of Retrieval-Augmented Generation

Indexing

Indexing typically encompasses partitioning heterogeneous textual data into finer-grained chunks [14]. These chunks are then transformed into vectors using Language Models (LMs) serving as the embedding model, enabling the computation of their semantic similarity to future queries [43, 63, 5, 28]. Finally, the index is constructed in which each initial text segment is paired with its corresponding embedding vector as key-value pairs and stored in a knowledge base [44, 59], facilitating fast and on-demand retrieval in subsequent stages [22, 14, 58]. This process is often conducted offline [15], and if the data source contains noise [8], certain preprocessing steps can be applied, such as stemming and removing stop words [15, 22].

Retrieval

Retrieval refers to encoding a question into a vector using the same embedding model employed during indexing, and then computing its similarity, typically using cosine similarity, as defined in Equation (2.6), with stored chunk vectors. The similarity value ranges from -1 (very dissimilarity) to 1 (very similarity), with 0 representing no similarity [5]. $\mathbf{a}$ and $\mathbf{b}$ denote their vectors. a_i and b_i represent their values in the i-th dimension [59, 28]. These embeddings are often compressed and projected into a space with fewer dimensions [5]. The top-k passages with the highest similarity scores are subsequently selected to serve as augmented contextual knowledge for the current query [15, 22, 58, 59]. As noted in Section 2.1.3, LM-based embedding models can utilize the advantages of contextual representations to identify semantic characteristics of both chunks and questions with improved performance [30], known as "dense retrieval" outperforming those non-contextual embeddings [15, 22, 5, 6], mentioned in Appendix A.2.

$$\text{Cosine Similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2} \sqrt{\sum_{i=1}^{n} (b_i)^2}} \text{[59]} \tag{2.6}$$

Generation

The question is augmented with retrieved content to form a prompt for the LLM to generate responses [22, 58]. If the history of previous conversations is available, it can also be incorporated into the instruction to enable conversational capabilities [15, 14]. While maintaining consistency and reliability with retrieved documents is difficult, it is also necessary to manage the trade-off between faithfulness to reference knowledge and creative generation [22].

2.4.2 Advancements in RAG Process Design

The earliest RAG architecture employed a retriever composed of two BERT models: one for generating vector representations of documents and the other for encoding questions, and a T5-like PLM as the generator, which required complex end-to-end fine-tuning [54, 5], also known as "joint training" [36]. LLMs can serve as generators, breaking this paradigm [15]. Although some closed-source LLMs only accept input, and output without permitting fine-tuning, they have fully acquired general language capabilities in parameters [1, 14]. Besides, many powerful embedding models have been developed specifically for RAG, rather than relying on general PLMs like BERT [64].

These advances enable RAG–LLM systems to deliver substantial gains across domain-specific applications without fine-tuning [14, 22, 35], by integrating parametric general knowledge encoded in weights [59] with non-parametric knowledge retrieved from external sources [15, 36, 58, 54, 5].

2.5 Related Work

The RAG–LLM framework has been increasingly adopted in a range of downstream engineering applications. AMGPT [5] presents a domain-expert RAG–LLM text generator, based on the Llama2-7B model, specifically designed for queries in metal AM. It supports academic querying on general knowledge of metal AM, leveraging an external database comprising about 50 research papers and books, and investigates the impact of different RAG-LLM hyperparameters on response quality. NANOGPT [6], built on the LLaMA3.1-8B-Instruct model, offers similar functions and follows a comparable process for evaluating hyperparameters, but focuses on the domain of nanotechnology. [31] maintains an external database of approximately 70 articles and utilizes GPT-4o-mini to predict parametric relationships in "machining, deformation and additive" manufacturing processes, with a particular emphasis on formula derivation.

Other studies have sought to improve the index structure used in RAG to enhance retrieval performance. [58] employs a vector-based Knowledge Graph (KG) as the RAG database to support queries about Smart Manufacturing (SM), but neither KG nor the query set is publicly available. [59] introduces a composite indexing strategy for RAG combining vectors, property graphs and keywords, and uses GPT-4o to conduct querying in the architectural domain. In terms of novelty, [63] manages the RAG database through subdirectory organization and incorporates customer preference adjustments into the RAG–LLM pipeline to handle queries related to construction management, while also evaluating the response quality of different LLMs.

In addition to encoding textual content from literature, [28] employs Vision Transformer (ViT) [12] to process images in those publications as part of the RAG database, deriving a conversational system for the materials science domain based on GPT-4o. [30] also adopts a multimodal RAG workflow, utilizing retrieved knowledge from research papers covering figures and text, and employs different Multimodal Large Language Models (MLLMs) for anomaly detection, classification, and interpretation generation in the Laser Powder Bed Fusion (L-PBF) manufacturing process.

Furthermore, FDM-Bench [13] proposes a benchmark corpus for evaluating the performance of LLMs on FDM-related knowledge, incorporating questions across several skill tiers in both multiple-choice and open-ended formats, along with a set of G-code samples indicating abnormal scenarios, thereby enabling a comprehensive assessment of models. LLM-3D [25] directly applies GPT-4o as an MLLM to recognize visible anomalies during 3DP through cameras and to automatically adjust process parameters, without using RAG, and notes potential discrepancies between MLLMs and human evaluations.

2.6 Research Gaps and Intended Project Contributions

The RAG-LLM solution is advancing but still at the initial phase [8, 15, 43]. In specialized domains, RAG deployment remains behind the progress of LLMs and real-world implementation is complex [44]. Many RAG optimization approaches exist in theory [22, 15], with little application in engineering practice

[44] and the impact of RAG on LLMs is not uniformly positive [8, 30]. However, applying RAG-LLM to AM provides a practical approach [43, 32] for accumulating engineering experience [15, 8]. The existing related literature is limited in quantity, but current studies could be further explored and improved.

Specifically, for resource-constrained RAG-LLM projects, evaluating final outputs and collecting data can be particularly challenging. Some studies have proprietary official datasets [30, 32]. Reference answers of open-ended questions for NANOGPT [6] were produced by PhD students and professors in the field. AMGPT [5] did not provide the ground truth, but its evaluation was likewise conducted by domain experts. In [63], both queries and ground truth were manually reviewed and improved by five engineers. However, FDM in AM lacks such resources, causing certain concerns. [28] did not even perform some necessary evaluations. Neither [6] nor [5] mentioned how literature data quality was assessed. And [58] did not publish complete data about its KG or evaluation questions.

Besides, to the best of our knowledge, there is **no** RAG-LLM querying or conversational system on AM that targets FDM using PLA. Though AMGPT focuses on metal AM, its scope is broad [5], as metal AM itself includes various aspects such as wire arc AM [10]. [58] focuses on SM but does not specify the domain it covers. In addition, there is currently **no** RAG-LLM product that can explicitly recommend industrial parameters [14] for manufacturing while mitigating hallucinations of LLMs. NANOGPT [6] includes only one query on parameter recommendation. FDM-Bench [13] is designed to evaluate LLMs' understanding of FDM but, despite including some PLA-related knowledge, it may ignore FDM parameter recommendations.

Also, the accelerated development and iteration of LLMs present challenges to engineers, who are interested in RAG but lack AI background [17, 38, 42, 44]. Specifically, some studies have investigated how different RAG-LLM hyperparameters or prompts affect output performance [5, 6], but **few** have addressed how outputs vary across different LLMs in RAG [31, 63, 30, 32, 59].

[63] organizes the RAG database with directories using only string labels, and since only GPT-4 can interpret this hierarchy, subsequent evaluations are limited to this model, constraining further exploration of LLMs. In addition, [58] utilizes a vector-based KG as the RAG database to interpret structured semantic relationships, but building and maintaining such a large KG is challenging for resource-constrained FDM. And based on listed prompts in the article, that study may only focus on semantic analysis without parameter recommendations.

Based on above potential research gaps, intended contributions of this project are as follows:

- Provide a comprehensive literature review of knowledge related to LLMs to help non-AI specialists quickly grasp recent developments in this field.
- Collect and **filter** literature about AM, specifically on FDM using PLA, to build a high-quality RAG database for academic QA in a conversational system with LLMs, thereby accumulating practical experience [31].
- Conduct a **preliminary** study on recommending [14] FDM parameters using RAG, considering PLA and its composites [49, 39], thus filling research gaps.
- Design questions for academic querying and FDM parameter recommendations [31], further contributing to the FDM-Bench evaluation corpus [13].
- Employ a state-of-the-art LLM-based evaluation approach to assess system performance [58], addressing constraints of limited evaluation resources. This is also referred to as "embedding-based metric" [27].
- Explore using RAG alone for storing structured data, rather than a KG, to find a more convenient solution [58].
- Enhance the RAG database structure through coding, instead of relying solely on LLMs to interpret the database folder hierarchy [63].
- Analyze output styles of different LLMs with RAG as a downstream application case, and provide possible feedback to domain researchers or LLM developers to guide adjustments in general-purpose or domain-specific training [7, 15].

Chapter 3

Research Methodology

This chapter outlines the project methodology. Section 3.1 introduces the process of data preparation. Next, different system components are explained in Section 3.2. Section 3.3 presents several adjustable system hyperparameters and Section 3.4 discusses employed evaluation approaches. Finally, experimental environment is summarized in Section 3.5.

3.1 Data Preparation

3.1.1 Literature Collection for RAG Database

FDM and PLA literature for the RAG database were collected using the backward snowballing. The entire process is visualized in Figure 3.1 and specific steps are outlined and explained below.

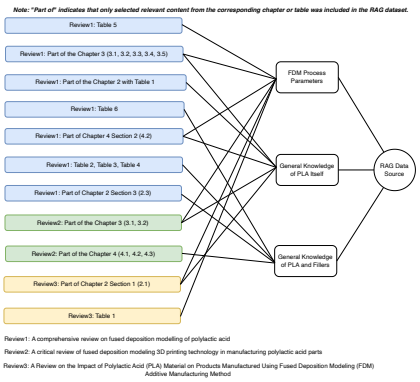


Figure 3.1: Visualisation of Literature Collection Process for RAG Database

1. A Google Scholar search with the keyword [28, 6] "Additive Manufacturing of Polylactic Acid via Fused Deposition Modeling" was used to select three review articles [49, 39, 29] as the starting point.
2. Backward snowballing was then performed based on section or chapter structures or tables in these three reviews, with referenced literature manually downloaded. Papers associated with each section or table were stored in separate folders [63].
3. The collected literature was further refined according to the content of these sections or tables, and organized into three categories: "FDM Process Parameters", "General Knowledge of PLA Itself" and "General Knowledge of PLA and Fillers", **each of which will serve as a separate indexing engine for RAG**.

Three review articles, listed in Table 3.1 were read and summarized thoroughly to grasp the knowledge structure of this domain. In brief, they are referred to as **Review1** [49], **Review2** [39] and **Review3** [29]. They all focus on FDM and PLA, without addressing other AM processes [10] or alternative materials such as Acrylonitrile Butadiene Styrene (ABS) [49, 39]. They collectively cover a wide range

of knowledge on the target topic, including PLA composites with a variety of fillers rather than being limited to carbon-based PLA.

Table 3.1: Three FDM PLA Review Articles

FDM PLA Review Article Title	Reference
Review1: A comprehensive review on fused deposition modelling of polylactic acid	[49]
Review2: A critical review of fused deposition modeling 3D printing technology in manufacturing polylactic acid parts	[39]
Review3: A Review on the Impact of Polylactic Acid (PLA) Material on Products Manufactured Using Fused Deposition Modeling (FDM) Additive Manufacturing Method	[29]

During snowballing, manual screening was applied. For example, some references that mentioned ABS-related knowledge or methods for analysing thermal properties of PLA, such as "DSC" and "HDT" with formula derivations, were considered outside the scope of this project. Conversely, some references on PLA composites relevant to process parameters were assigned to both "FDM Process Parameters" and "General Knowledge of PLA and Fillers".

Besides, given the volume of publications in AM has been doubling approximately every two years [28], such folder-based management is required. Typically, literature for RAG datasets should be collected through searches, followed by expert screening [8]. This project used backward snowballing in place of expert screening. The data quality in this case is assumed to be acceptable [6, 31, 5, 43] since all literature is accessible via Google Scholar [28].

In total, 210 unique articles (excluding Review3) were collected after removing duplicates, serving as **unstructured text data** [15] in PDF format. The number of articles in each of the three folders is listed in Table 3.2. Review1 and Review2 were placed in all three folders, while Review3 was excluded and *the reason is explained in Section 3.1.4 in italics.*

Table 3.2: Number of Articles in Each of the Three Folders for RAG Database Organisation

Folder Name	Number of Articles
FDM Process Parameters	124
General Knowledge of PLA Itself	101
General Knowledge of PLA and Fillers	94

3.1.2 Literature Data Naming Management

Literature PDF files downloaded from Google Scholar often have disordered alphanumeric filenames. An initial approach was to rename each file using its title; however, some titles exceeded the system length limit. In this project, each distinct document was assigned a unique identifier like "`reference_1.pdf`". The literature in three folders was then updated using these identifiers.

3.1.3 Table Data for Parameter Recommendations

In the later optimization stage of this project, a tabular dataset in HTML format was manually created as structured data [15, 44], addressing parameter recommendations, listed in Appendix A.3. In RAG, it is also known as Internal Data Augmentation (IDA), maximizing the use of knowledge already in documents [22]. The information in table was sourced from Review1 and [18], with manual verification.

3.1.4 Question Set Design

The question set was divided into **open-ended QA** and **FDM parameter recommendations**.

Open-Ended Question Answeing

A total of 30 open-ended queries [13, 59] were designed and grouped into three categories: FDM technology, properties and applications of PLA, and PLA composite knowledge, as summarised in Table 3.3.

Table 3.3: Counts of Open-Ended QA Queries by Category

Question Category	Number of Questions
FDM Technology	12
Properties and Applications of PLA	9
PLA Composite Knowledge	9
Total	**30**

These questions were derived from summaries of Review1 and Review2. Rather than masking a sentence-level fact and asking about it directly, this project formulated questions more abstractly to discourage direct lexical matching, thereby reducing reliance on chunks from original review articles. For example, the question "What are the future development trends of polylactic acid (PLA) composites?" prompts the system to summarise across multiple sources rather than retrieve a single sentence from chunks.

FDM Parameter Recommendations

In terms of parameter recommendations, Review2 focuses more on describing the relationship between properties of printed products and FDM parameters, without specifically introducing parameters themselves. *Review3 provides a thorough summary of FDM parameters, but some parameter names were uncommon, absent from Review1 and Review2, and some were overly similar, such as "print speed" and "printing speed".* Consequently, this project extracted five FDM parameters: "nozzle temperature", "printing speed", "layer height", "bed or platform temperature" and "nozzle diameter", distilled from the comparative analysis of all three reviews. Besides, based on Review1 and Review2, 19 different PLA–filler material combinations were identified, primarily involving carbon-based, plant-based and mineral-based fillers, which also represent three main approaches in PLA composites. Together with pure PLA, a total of 100 ((19+1)*5) parameter recommendation questions were formulated.

3.2 System Implementation

3.2.1 Embedding Model Used in Project

In this project, "`BAAI/bge-large-en-v1.5`"[1], a variant of the BAAI General Embeddings (BGE) model, was used as the embedding model to transform segmented chunks and user queries into fixed 1024-dimensional vectors for similarity calculation, this dimensionality being determined by the model architecture and not subject to modification. The BGE model family is the state-of-the-art encoder [63], mapping text into contextual embeddings [64], using a large open-source corpus of QA pairs tailored for retrieval tasks [63]. It has been downloaded over 20 million times on HuggingFace, ranking it among the most widely adopted embedding models in the field [64].

3.2.2 Large Language Models Used in Project

LLMs used in this project include GPT-4o, GPT-4.1, DeepSeek-V3 and DeepSeek-R1. They exhibit different characteristics. The OpenAI GPT family is a pioneer among LLMs [27]. GPT-4o supports multimodal input and output [23], while GPT-4.1, as an upgraded version of GPT-4, demonstrates strong performance in text processing[2]. Both are closed-source. The DeepSeek family has gained global recognition for offering performance close to GPT models at a cost significantly lower than competitors. It mainly comprises two products: V3 for chat and R1 for reasoning[3]. Both are open-source [44] but they are accessed via APIs in this project. In addition, GPT-4o was specifically employed to answer parameter recommendation questions using the table data.

3.2.3 Prompt Engineering

The prompt engineering in this project was designed to integrate widely adopted solutions. The **base prompt** of this project is listed in Appendix A.4. Its content was inspired by the prompt template in [44]. The prompt design followed three components:

1. Assign the LLM a task-specific role [13].
2. Introduce concise background information and explicitly state the task objectives [13, 63].
3. Provide reasonable and operational instructions [13, 63].

[1] https://huggingface.co/BAAI/bge-large-en-v1.5
[2] https://openai.com/index/gpt-4-1/
[3] https://api-docs.deepseek.com/

In particular, the LLM was required to avoid depending on its general encoded information when generating answers [31], and if the answer is unknown even with retrieved materials, it should refuse to respond. Additionally, it was instructed to indicate the source of the retrieved content [15], thereby mitigating hallucinations [44].

3.2.4 RAG Implementation

The RAG workflow was implemented using functions provided in `LlamaIndex` library[4], an open-source framework for building applications with RAG and LLMs [5, 15, 22, 44]. It consisted of the following steps:

1. **Chunking** – Using `SimpleDirectoryReader()`, each page of every document was first cached. Then, `SentenceSplitter()` was applied to partition or parse literature pages or tabular data into chunks.
2. **Embedding and Indexing** – Using `VectorStoreIndex()`, each chunk was converted into a contextual embedding vector [15] and indexed in the retrieval engine [44, 6, 5].
3. **Local Storage** – The resulting vectors with index were stored locally using `storage_context.persist()`.
4. **Loading Existing Embedding with Index** – When previously created vectors with index were available, they could be directly loaded via `load_index_from_storage()`, avoiding redundant computation.

Through `LlamaIndex`, RAG integrated the embedding model and LLM APIs, and passed the prompt to LLMs for semantic alignment [58], thereby establishing the core functional architecture. In addition, two optimizations were implemented.

Metadata Attachment

Metadata such as file name and type were added to each page through the `Document` class in `LlamaIndex` so that, after parsing, every chunk could explicitly identify the specific reference and page from which it originates [15], also serving as a form of IDA [22].

Expert and Global Engines

To manage literature data, the "expert engine" was built for each folder, and a selector was provided to choose the desired engine for users. The table data was also used as a separate engine. LLMs then respond based on retrieved content of files from the selected folder, with a dedicated prompt containing the folder name passed to it: "*You are an intelligent conversational assistant for additive manufacturing based on Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). You specialize in answering user questions strictly based on the content of documents retrieved from the 'folder_name' folder.*" The whole prompt is listed in Appendix A.5.

In addition, a hash-based deduplication process by `hashlib.md5()`[5] was applied to all literature pages, and the resulting unique set was used to construct a global engine. It contains all unique pages from the literature with the base prompt listed in Appendix A.4. During inference, the response included an additional prefix indicating whether it was generated using the global engine or a specific expert engine.

3.2.5 Frontend and Backend

Considering the trend in manufacturing toward user-oriented interactive approaches [35], the system included both frontend and backend components, allowing a local computer to simulate server operations. As shown in Figure 3.2, the frontend interface allowed customers to browse and download documents in the current index, switch between expert engines or the global engine (All Documents), and see an ethical notice, before starting, that conversation history would be collected. The frontend was built with JavaScript (Vue 3[6]) and HTML, with CSS used for simple color schemes and layout design. The backend was developed using the Python `FastAPI` library [44].

[4] https://www.llamaindex.ai/
[5] https://docs.python.org/3/library/hashlib.html
[6] https://vuejs.org/

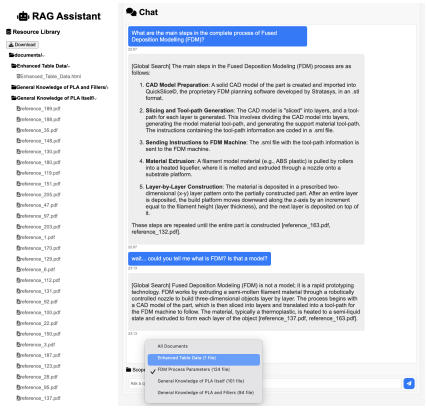


Figure 3.2: Frontend Interface

3.2.6 Logging

To support scientific experimentation, a comprehensive `logging` mechanism was implemented [44]. It recorded various events, such as system states and index creation times, and stored conversations in JSON files, using UUIDs [63] for identification.

3.3 Adjustable Hyperparameters

3.3.1 Chunk and Overlap Sizes

Each chunk was generated with a fixed size [58, 31, 59, 63] of 512 tokens. This choice aligns with the optimal configuration for the embedding model in best-practice scenarios[7], and represents a compromise between excessive fragmentation and the inclusion of unrelated knowledge from overly long text segments [28]. In addition, an overlap [44] of 50 tokens between chunks [15] was applied to reduce the loss of relevant context caused by sentence breaks, enhancing retrieval performance [44]. The effect of varying overlap values is further examined in subsequent experiments.

3.3.2 Sampling Temperature

The sampling temperature regulates the randomness and creativity of LLM generations [5]. Lower temperatures (below 0.3) [6] may cause models to depend excessively on retrieved content, yielding responses that simply replicate retrieved materials [15], whereas higher temperatures (above 1) tend to output indistinct or hallucinated content. A balance should be maintained between accurate factual retrieval and the creative integration of insights across multiple chunks [5, 7]. Therefore, in this project, a sampling temperature of 0.5 was used for open-ended QA, while a temperature of 0.3 [6, 31] is adopted for parameter recommendations.

3.3.3 Output Token Limit

The output token limit specifies the largest number of tokens for each LLM output. As noted in [6, 5], lower limits often result in brief answers, while higher limits may cause redundant output. This project slightly extended the common industry setting of 768 tokens to 1024 tokens for both QA and parameter recommendations.

3.3.4 Top-K Retrieval

The "Top-K" parameter determines the number of chunks retrieved by RAG that are the most relevant to the query and used as augmented content, based on cosine similarity. A lower Top-K, such as 2, may retrieve limited relevant knowledge, whereas a higher value, such as 5, probably brings in unrelated content. In this project, a Top-K value of 3 was used for both QA and parameter recommendations [5, 6, 44].

[7] https://huggingface.co/BAAI/bge-large-en-v1.5

3.4 Evaluation Approaches

3.4.1 LLM-Based Evaluation

This project employed an LLM-based quantitative evaluation method [58], implemented using **DeepSeek-V3**. With tailored prompts [15], the LLM is guided to emulate human judgment, grasp the task in context, and assess RAG–LLM generations. This evaluation is conducted across multiple aspects and aligns closely with expert assessments, even without the ground truth [27]. Five evaluation dimensions, as listed below, were designed with scores ranging from 1.0 to 10.0 (allowing decimal values). Their definitions were also included in the prompt, together with sufficient contextual information to minimize ambiguity. The prompt for evaluation, inspired by [58], is listed in Appendix A.7.

- **RAG-Related Dimensions:**

  1. **Answer Relevance (AR)** – The degree to which the content of the answer is relevant to the question [15, 22, 59].
  2. **Context Precision (CP)** – The proportion of knowledge in the response that is relative to the query, relevant to all knowledge provided in the response. It assesses whether the evidence retrieved by RAG is effectively integrated into the response to address the query, thereby indicating the response quality [58, 15, 22].
  3. **Completeness** – Whether the answer is complete, for example, whether there are any incomplete sentences [27].

- **LLM-Related Dimensions:**

  4. **Coherency** – Whether the answer is smooth and clear, for example, how well the sentences are connected to each other. Some literature refers to this as fluency [27, 22], but according to dictionary definitions[8], "fluency" tends to describe the smoothness of spoken language rather than the logical coherence between sentences.
  5. **Formatting Quality (FQ)** – Evaluate the use of Markdown. Consider whether Markdown syntax is present and whether the formatting meaningfully improves readability and organization [35].

Explanation dimension was also designed, where the LLM was prompted to justify assigned scores and to identify potential issues in answers if possible [5]. Some studies have suggested evaluating capabilities of LLMs in "negative rejection" and "information integration". Negative rejection refers to the ability of LLMs to refuse to answer when retrieved materials cannot address the query. This can actually be configured through the prompt and assessed by observing generated outputs. Information integration denotes the ability of LLMs to synthesize knowledge from multiple useful retrieved passages [8, 22, 15], and can be evaluated via **AR** and **CP**.

When using DeepSeek-V3 as the evaluator, the temperature was set to 0.1 to enhance direct judgment, and the maximum output token limit was set to 2000 to provide sufficient space for the "Explanation" dimension.

3.4.2 Latency

Latency is also a key metric for assessing RAG-LLM systems, as it is particularly crucial for interactive scenarios. It measures the duration required for the system to produce a reply to a question, reflecting both the retrieval speed of the RAG and the inference speed of LLMs [58, 63].

3.4.3 Metrics for Parameter Recommendations

During the later optimization phase of this project, a set of quantitative metrics was introduced for parameter recommendation tasks based on the table data, namely data coverage rate, hit rate, precision and recall [15, 16].

[8] https://dictionary.cambridge.org/

Data Coverage Rate

As shown in Equation (3.1), the Data Coverage Rate represents the proportion of questions that can be answered based on the data, regardless of RAG or LLM configurations.

$$\text{Data Coverage Rate} = \frac{\text{Number of questions answerable from the data}}{\text{Total number of questions}} \tag{3.1}$$

Hit Rate

As shown in Equation (3.2), the Hit Rate measures, among questions answerable from the data, the proportion for which the model provides an answer based on retrieved content without refusal [15, 59].

$$\text{Hit Rate} = \frac{\text{Number of answered questions from retrieved content}}{\text{Number of questions answerable from the data}} \tag{3.2}$$

Precision

As defined in Equation (3.3), precision denotes the proportion of predicted positive instances that are actually positive. True Positives (TPs) are instances that are predicted as positive and are actually positive. False Positives (FPs) are instances that are predicted as positive but are actually negative. Precision reflects the ability of the model when predicting an instance as positive [16].

$$\text{Precision} = \frac{TP}{TP + FP} [16] \tag{3.3}$$

In the context of parameter recommendations, TP refers to the model providing a correct answer for a question based on retrieved content [59]. FP refers to the model providing an incorrect answer for such a question. The definition of task-specific precision is given in Equation (3.4).

$$\text{Precision}_{\text{task}} = \frac{\text{Number of correct answers from retrieved content}}{\text{Number of answered questions from retrieved content}} \tag{3.4}$$

Recall

As defined in Equation (3.5), recall denotes the proportion of TP instances that are correctly predicted as positive by the model. False Negatives (FNs) are instances that are predicted as negative but are actually positive. Recall reflects the ability of the model to capture all relevant positive instances [16].

$$\text{Recall} = \frac{TP}{TP + FN} [16] \tag{3.5}$$

In the context of parameter recommendations, FN refers to cases where the model **refuses** to answer a question where the required correct data can actually be retrieved [59]. The definition of task-specific recall is given in Equation (3.6).

$$\text{Recall}_{\text{task}} = \frac{\text{Number of correct answers from retrieved content}}{\text{Number of questions answerable from the data}} \tag{3.6}$$

3.5 Experimental Environment

All project code files and experiment logs (results) are available on GitHub[9]. The project was conducted using Python 3.9, with all required Python packages listed in `requirements.txt`. As noted earlier, the RAG was built with LlamaIndex[10], and the embedding model was downloaded from Hugging Face[11]. Four LLMs (DeepSeek-V3, DeepSeek-R1, GPT-4o and GPT-4.1) were accessed via official APIs[12] [13]. GPU experiments were run on the Featurize[14] server, while the local environment was an Apple M1 Pro machine (16GB, macOS 14.7.6) with the Metal Performance Shaders[15] (MPS).

[9] https://github.com/ZihaoChen189/AM_Chat_System_based_on_RAG_and_LLMs
[10] https://www.llamaindex.ai/
[11] https://huggingface.co/
[12] https://api-docs.deepseek.com/
[13] https://platform.openai.com/docs/overview
[14] https://featurize.cn/
[15] https://huggingface.co/docs/accelerate/usage_guides/mps

Chapter 4

Results Analysis

This chapter analyzes experiment results, including both quantitative and qualitative assessments. The main experiment procedure is explained in detail in Section 4.1.

4.1 Overview of Main Experiments

This project involves two question sets, as introduced in Section 3.1.4:

1. 30 open-ended queries were answered using the global engine, which retrieves from the full literature corpus.
2. 100 parameter recommendation questions were answered initially using the expert engine "FDM Process Parameter".

However, among 100 parameter recommendation questions, GPT-4o was able to answer only 15, while refusing to respond to the remaining 85 due to a lack of relevant knowledge in the retrieved information for specific parameter recommendations. For example, "*I do not know. The retrieved documents do not provide information about the appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC).*" Those remaining 85 parameter recommendations therefore require the table data as an additional expert engine, enabling the system to produce responses.

On the one hand, answers to all 45 questions, comprising 30 open-ended queries and 15 directly answerable parameter recommendation questions, **all of which can be answered using literature textual data**, were assessed using the LLM-based evaluation, as described in Section 3.4.1. Notably, those 15 parameter recommendation questions were also treated as a subset of the open-ended questions. Although these queries were in an open-ended format, the system, using four LLMs, successfully retrieved relevant knowledge and provided corresponding answers. The result analysis for these **directly answerable questions** based on literature textual data is presented in Section 4.2.

On the other hand, 85 parameter recommendation questions that became answerable with GPT-4o after incorporating table data were evaluated using the defined quantitative metrics, as described in Section 3.4.3. The result analysis for these **initially unanswerable parameter recommendation questions** is presented in Section 4.3.

The experimental configurations are summarized in Table 4.1.

Table 4.1: Hyperparameter Configurations for Different Tasks

Task Type	Chunk Size	Overlap	Sampling Temperature	Output Token Limit	Top-K Retrieval
30 Open-Ended Academic Queries	512	50	0.5	1024	3
15 Parameter Recommendation Questions (directly answerable, open-ended subset)	512	50	0.3	1024	3
85 Parameter Recommendation Questions (answered by table data)	512	200	0.3	1024	3

4.2 Results of Open-Ended Queries

The performance of different LLMs on the open-ended 45 questions was assessed using identical hyperparameters specified in Table 4.1, together with the fixed prompt. This setup guarantees that observed result variations resulted from intrinsic abilities of RAG-LLMs instead of changes in hyperparameter

settings [13]. Notably, the set of 45 open-ended queries includes 15 directly answerable parameter recommendation questions, whose answers can all be retrieved from literature textual data. These questions were evaluated in the same manner as the other open-ended queries.

For each LLM, the average score across all questions was calculated separately for each evaluation dimension, and this procedure was repeated three times [58] to account for API randomness in the LLM evaluator [26]. Detailed results from three trials are presented in Appendix A.8. For each LLM and each dimension, the mean across three trials was used as the final score, as shown in Figure 4.1 with the y-axis starting from 7.

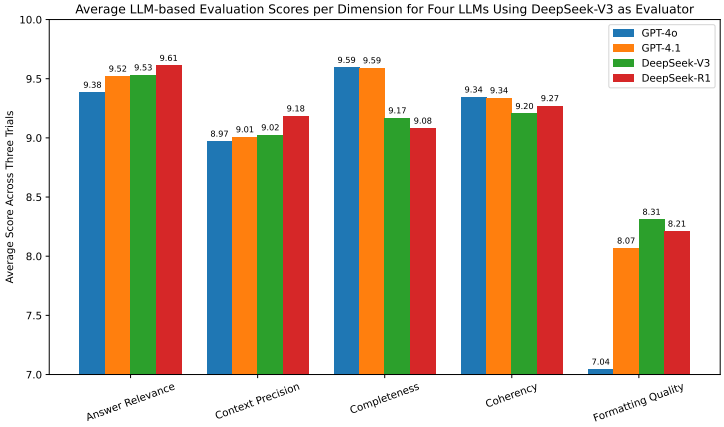


Figure 4.1: Average LLM-Based (DeepSeek-V3) Evaluation Scores per Dimension for Four LLMs

4.2.1 Quantitative Analysis

LLMs exhibit similar patterns in AR and CP dimensions. Specifically, DeepSeek-R1 achieves the best performance, with a score of 9.61 in AR and 9.18 in CP. It is followed by GPT-4.1 and DeepSeek-V3, which produce comparable second-best results. GPT-4o scores relatively lower, obtaining only 9.38 in AR and 8.97 in CP. For the Completeness dimension, GPT-4.1 and GPT-4o achieve the highest score of 9.59. DeepSeek-R1 obtains the lowest score at 9.08, representing a substantial gap from the top-performing models, while DeepSeek-V3 performs slightly better with a score of 9.17. In the Coherency dimension, GPT-4.1 and GPT-4o have the top score of 9.34. DeepSeek-V3 has the lowest value at 9.20, whereas DeepSeek-R1 attains a higher score of 9.27. Results for the FQ dimension are noteworthy: GPT-4o receives a notably low score of 7.04, followed by GPT-4.1 at 8.07 and DeepSeek-R1 at 8.21. DeepSeek-V3 achieves the highest score of 8.31 on FQ.

4.2.2 Qualitative Analysis

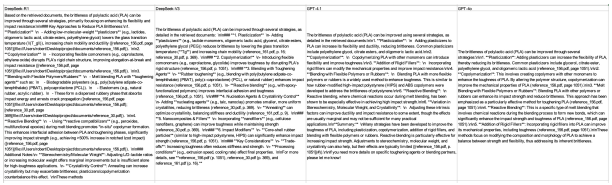
Answer Relevance and Context Precision

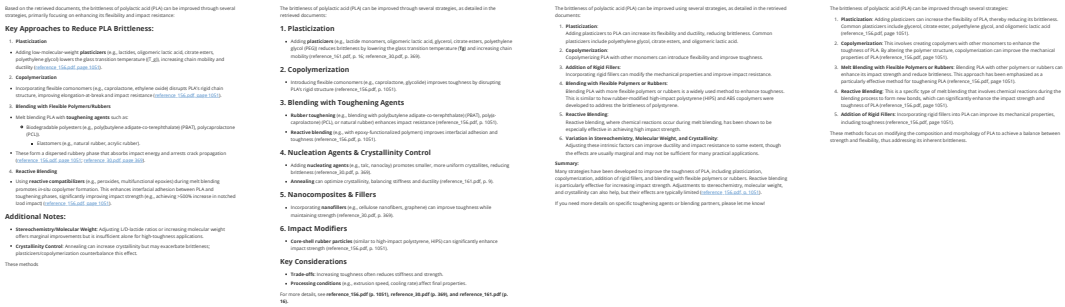
Generally high scores in AR indicate that all LLMs are able to generate responses closely aligned with given questions using the retrieved information. At the same time, when considering consistently high scores in CP, these results also provide evidence for the effectiveness of converting literature into vectors using the embedding model and retrieval based on cosine similarity. DeepSeek-R1 demonstrates strong performance in both AR and CP, possibly due to its cold-start training on high-quality reasoning data [17]. Such training may have equipped it with robust reasoning capabilities, enabling it to integrate retrieved literature more logically and produce responses that are highly relevant to questions, suggesting a strong capacity for information integration [8]. DeepSeek-V3 has slightly higher scores than GPT-4.1, which may be attributed to its knowledge distillation from the reasoning ability of DeepSeek-R1 [21]. Considering the specialized MoE of DeepSeek-V3 [38] and the closed-source nature of GPT-4.1 [26], conducting a critical analysis based on their model sizes is not possible. In contrast, the relatively lower score of GPT-4o may be due to its design focus on balanced multimodal input and output [23], which leads to slightly weaker text-based information integration.

Completeness

Although scores of these models generally fluctuate within a limited range, the performance of DeepSeek-R1 in the Completeness dimension appears to be an outlier. Manual examination of its outputs revealed that one question was left unfinished, likely due to truncation caused by reaching the output token limit. For that question, outputs from all four LLMs were collected and are presented in Table 4.2 (with identical row and column sizes from Excel). As outputs contained extensive Markdown syntax in a specific HTML format, *simple* escape character processing was applied, and compiled Markdown results are shown in Figure 4.2 (at the same scaling ratio). The order of models presented in Table 4.2 and Figure 4.2 is consistent.

Table 4.2: Raw Outputs of Four LLMs on One Question Under Completeness Dimension





(a) DeepSeek-R1 (b) DeepSeek-V3 (c) GPT-4.1 (d) GPT-4o

Figure 4.2: Compiled Outputs of Four LLMs on the Same Question Under the Completeness Dimension

In Table 4.2, the amount of raw output decreases from DeepSeek-V3 to GPT-4o, and this is accompanied by a corresponding reduction in the length of compiled Markdown content shown in Figure 4.2. However, DeepSeek-R1 appears to deviate from this potential pattern, producing more raw output but displaying less compiled Markdown content compared to DeepSeek-V3. This probably indicates that DeepSeek-R1 allocates more tokens to certain Markdown tags.

Furthermore, other models typically list five to six concise points for this question, whereas DeepSeek-R1 elaborates more on each, such as in “Blending with Flexible Polymers/Rubbers”. This may also stem from the unique mechanisms of DeepSeek-R1 [17]: it reasons over the retrieved information with the query to produce high-quality answers, while generating extensive Markdown tags for User Experience (UX) [19], which may cause a truncation. This also aligns with the reason provided by the LLM evaluator: “The answer is nearly complete but ends abruptly with ’These methods’, suggesting a minor truncation or missing conclusion”.

Coherency

In terms of Coherence, four LLMs differ little. GPT series may produce smoother and clearer responses than DeepSeek series. A possible explanation, from the data perspective, is that OpenAI is an English-based development team. While these LLMs are all capable of multilingual interaction to some extent, GPT series may be more likely to learn higher-quality English-language corpus than DeepSeek series, which are developed in a Mandarin context. DeepSeek-R1 slightly outperforms DeepSeek-V3, possibly because its stronger logical structure in content offsets limitations in training data.

Formatting Quality

For clear analysis, the conversation history of different LLMs on three *answerable* parameter recommendation questions is presented in Table 4.3.

Table 4.3: Raw Outputs of Four LLMs on Three Questions Under Formatting Quality Dimension

Questions	GPT-4o	GPT-4.1	DeepSeek-V3	DeepSeek-R1
What is the appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with continuous carbon fibre (CCF)?	The appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with continuous carbon fibre (CCF) is 1.5 mm, as mentioned in the document \"reference_124.pdf\" from the FDM Process Parameters folder.	The appropriate nozzle diameter for 3D printing using fused deposition modeling (FDM) with polylactic acid (PLA) mixed with continuous carbon fibre (CCF) is 1.5 mm. In the referenced study, a stainless steel nozzle with a diameter of 1.5 mm was used for extruding PLA reinforced with continuous carbon fibre (CCF) on a standard Prusa i3 MK3S 3D printer. This larger nozzle size facilitates the smooth feeding and impregnation of the continuous carbon fibre with the molten PLA, minimizing the risk of clogging and ensuring proper composite formation [reference_124.pdf, page 3].	Based on the retrieved documents, the appropriate nozzle diameter for 3D printing PLA mixed with continuous carbon fiber (CCF) is **1.5 mm**. This information comes from a study where stainless steel nozzles of this diameter were used to print PLA-CCF and PLA-SCF-CCF composite specimens with a standard Prusa i3 MK3S 3D printer (reference_124.pdf, page 3). \n\nThe document specifically states: \n*\"Stainless steel nozzle having the diameter of 1.5 mm was used. The same process was used to print PLA and PLA-SCF specimens with standard Prusa i3 MK3S 3D printer.\"* \n\nNo other nozzle diameters are mentioned for CCF-reinforced PLA in the retrieved documents. For unreinforced PLA, nozzle diameters of 0.2 mm and 0.4 mm are mentioned (reference_112.pdf, page 289), but these are not specified for CCF composites.	Based on the retrieved documents, the appropriate nozzle diameter for 3D printing PLA mixed with continuous carbon fiber (CCF) is **1.5 mm**. This is specified in the experimental setup where:\n \n>*\"Stainless steel nozzle having the diameter of 1.5 mm was used. The same process was used to print PLA and PLA-SCF specimens with standard Prusa i3 MK3S 3D printer.\"* \n**Source**: 'reference_124.pdf', page 3 (FDM Process Parameters folder). \n\nNo other nozzle diameters (e.g., 0.2 mm or 0.4 mm mentioned in other documents) were referenced for CCF-reinforced PLA printing.
What is the appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF)?	The appropriate nozzle diameter for 3D printing using fused deposition modeling with PLA mixed with short carbon fiber (SCF) is 1.5 mm, as mentioned in the document reference_124.pdf.	The retrieved documents do not specify a particular nozzle diameter exclusively for 3D printing with polylactic acid (PLA) mixed with short carbon fibre (SCF). However, in the referenced study, both PLA and PLA-SCF specimens were printed using a stainless steel nozzle with a diameter of 1.5 mm on a standard Prusa i3 MK3S 3D printer. This suggests that a 1.5 mm nozzle is appropriate for printing PLA mixed with short carbon fibre, as it helps prevent clogging and ensures smooth extrusion of the composite material [reference_124.pdf, page 3].	The retrieved documents do not explicitly specify the nozzle diameter for 3D printing **PLA mixed with short carbon fiber (SCF)**. However, the following relevant details are available: \n\n1. **For PLA-CCF (continuous carbon fiber)**: A **1.5 mm nozzle** was used (reference_124.pdf, page 3). \n2. **For unreinforced PLA**: Nozzle diameters of **0.2 mm and 0.4 mm** were used (reference_112.pdf, page 289). \n\nSince the document mentions that the **same process** (with a 1.5 mm nozzle) was used for both **PLA-SCF-CCF** (short + continuous carbon fiber) and **PLA-CCF**, it is possible that a **larger nozzle (1.5 mm)** is preferred when carbon fibers (short or continuous) are involved to avoid clogging. However, the exact diameter for **PLA-SCF alone** is not explicitly stated. \n\nWould you like me to check if there are additional documents that might clarify this?	Based on the retrieved documents, the appropriate nozzle diameter for 3D printing PLA mixed with **short carbon fiber (SCF)** is **1.5 mm**. This is explicitly stated in the experimental setup for printing PLA-SCF composites: \n\n>*\"Stainless steel nozzle having the diameter of 1.5 mm was used. The same process was used to print PLA and PLA-SCF specimens with standard Prusa i3 MK3S 3D printer.\"* \n**Source**: 'reference_124.pdf', page 3 (FDM Process Parameters folder). \n\nNo other nozzle diameters (e.g., 0.2 mm or 0.4 mm referenced in other documents for pure PLA) are mentioned for SCF-reinforced PLA in the context of this document. The 1.5 mm diameter was used to accommodate the carbon fiber reinforcement while avoiding clogging.
What is the appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with carbon nanotube (CNT)?	The appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with carbon nanotubes (CNT) is between 177 and 185°C, as used in the tests described in the research paper [reference_114.pdf].	For 3D printing using fused deposition modeling (FDM) with polylactic acid (PLA) mixed with carbon nanotubes (CNT), the appropriate nozzle (extrusion) temperature is between 177°C and 185°C. This temperature range was used in tests for PLA/MWCNT (multi-wall carbon nanotube) nanocomposites and yielded promising results for 3D printing applications [reference_114.pdf, page 875].	Based on the retrieved documents, the appropriate nozzle temperature for **3D printing PLA mixed with carbon nanotubes (CNTs)** i s **between 177°C and 185°C**. \n\nThis information comes from a study where a **PLA/MWCNT (multi-wall carbon nanotube) nanocomposite** was tested for 3D printing (FFF technology). The extrusion temperatures used were: \n- **177°C to 185°C** (reference_114.pdf, page 875). \n\n### Additional Notes: \n- For **pure PLA**, the optimal nozzle temperature is typically **190–230°C**, with **215°C** often recommended for best mechanical properties (reference_188.pdf, page 784). \n- However, when **CNTs are added**, the required extrusion temperature may be slightly lower (as seen in the 177–185°C range). \n\nWould you like details on other parameters (e.g., printing speed, bed temperature) for PLA-CNT composites?	Based on the retrieved documents, the appropriate nozzle temperature range for 3D printing PLA mixed with **carbon nanotubes (CNTs)** is **177–185°C**. This temperature range was specifically tested for PLA/MWCNTs (multi-wall carbon nanotubes) nanocomposites with 0.5% and 1% CNT weight content. The extrusion at these temperatures resulted in successful 3D printing using FFF technology. \n\n>*\" The extrusion temperatures used in tests were between 177 and 185°C... [for] a PLA/MWCNTs nanocomposite... [with] promising result for the application... into 3D printing.\"* \n**Source**: 'reference_114.pdf', page 875 (FDM Process Parameters folder). \n\nThis range is notably lower than typical pure PLA printing temperatures (e.g., 190–230°C, as noted in 'reference_188.pdf'), likely due to the enhanced thermal properties imparted by CNTs. No higher temperatures were tested or recommended for PLA/CNT composites in the retrieved context.

Without constraints on output content or format, from results of parameter recommendation questions, GPT-4o tends to produce concise and directly relevant responses to a given question, but rarely incorporates Markdown formatting. This phenomenon is also consistent with outputs for QA shown in Figure 4.2.

This may still be related to its design, emphasizing compatibility in multimodal problem-solving rather than purely text-based optimization [23]. Similarly, GPT-4.1 does not actively employ Markdown tags. In contrast, DeepSeek series consistently wrap outputs in Markdown and actively integrate retrieved knowledge, whether in QA as shown in Figure 4.2 or giving such parameter recommendations. The slightly lower score of R1 compared with V3 may be partly attributable to its truncated responses illustrated before.

Moreover, DeepSeek-V3 often ends responses by asking whether the customer has further questions, consistent with the stated intention of DeepSeek for V3 to function as a conversational agent[1]. The prompt to LLMs advises follow-up questions only when a query is unclear, yet all questions in this project are explicit. Therefore, in terms of UX, DeepSeek-V3 is likely to be the most satisfactory among these LLMs, aligning with the manufacturing industry's current call for strengthening the customer-oriented interactive paradigm [35].

4.3 Results of FDM Parameter Recommendation Questions

In the initial experimental phase, literature textual data alone enabled the system to answer only 15 out of 100 parameter recommendation questions. The remaining 85 required the use of an additional HTML table as a separate expert engine for completion. Critically, this outcome demonstrates the prompt engineering described in Appendix A.5 effectively guides the LLM to perform negative rejection [8], declining questions that are not well supported by the provided literature, enhancing system interpretability and reliability [31, 15]. Using metrics defined in Section 3.4.3 and specific configurations shown in Table 4.1, results for assessing 85 FDM parameter recommendation questions are presented in Table 4.4.

Table 4.4: Evaluation Results for 85 Parameter Recommendation Questions Using Specified Metrics

Metric	Data Coverage Rate	Hit Rate	Precision	Recall
Value	0.61	1.00	1.00	1.00

4.3.1 Quantitative Analysis

Table 4.4 shows the data coverage rate is 0.61, indicating that, among 85 parameter recommendation questions, the table data contains useful parameter information for about 61% of them. The hit rate, precision and recall are all 1.0, demonstrating that, if the required information is available in the table data, the system can generate correct answers without errors.

4.3.2 Qualitative Analysis

The constructed table data covers the majority of the required information for parameter recommendation questions, with a data coverage rate of 61%, which is reasonable given the inherent incompleteness

[1] https://api-docs.deepseek.com/

of published literature. For instance, [49] provides parameters such as bed platform, layer height, nozzle temperature, and nozzle diameter for mixing PLA and CNC, but does not specify printing speed. Therefore, the absence of certain data in the table results from gaps in the source literature, rather than a failure of the RAG system to retrieve information. Compared to the other metrics, the relatively low data coverage rate also **alleviates concerns that parameter recommendation questions were specifically tailored to the table data**. Furthermore, the high hit rate, precision and recall indicate that the RAG system, using the embedding model [64], effectively encodes structured tabular data and matches knowledge to queries.

In addition, there remains potential ambiguity regarding certain answers. For example, the table does not provide a bed temperature recommendation for pure PLA, but when asked, the system offered the value used for PLA composites with other fillers, rather than refusing to answer. In this project, this response was considered acceptable, since given both the question and the table, a person without relevant background knowledge might reasonably adopt a similar approach.

4.4 Latency Analysis of Different LLMs

For each LLM, this project utilized the global engine (excluding table data) and recorded the total time required to answer three questions. Each experiment was conducted three times, and **the average time required to answer all three questions** was used as the final latency metric, as reported in Figure 4.3. Specific questions and detailed timing data are listed in Appendix A.9.

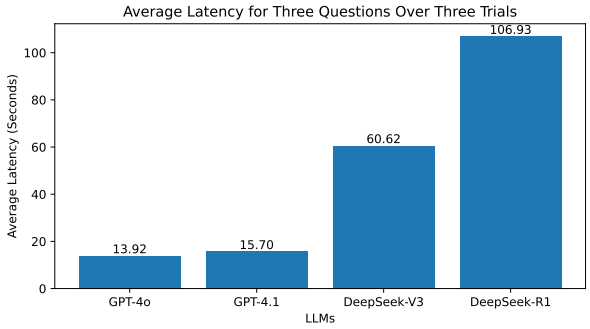


Figure 4.3: Average Latency of Different LLMs for Three Questions Over Three Trials

4.4.1 Quantitative Analysis

GPT series show notably lower average latency compared to DeepSeek series in the context of answering three questions. Specifically, GPT-4o achieves the lowest latency at 13.92 seconds, while GPT-4.1 is slightly higher at 15.70 seconds. In contrast, the latency of DeepSeek-V3 is about four times that of GPT-4o, and DeepSeek-R1 is even slower than DeepSeek-V3. The latency differences among these LLMs are clear, indicating that model selection significantly affects the efficiency of RAG-LLM systems, particularly in scenarios where real-time interaction is prioritized.

4.4.2 Qualitative Analysis

GPT-4o demonstrates outstanding speed, and as a multimodal model, it is likely optimized with various mechanisms for faster inference. Although GPT-4.1 is slightly slower, its performance on AR and CP is similar to DeepSeek-V3, making its text processing capability acceptable. In contrast, DeepSeek models are considerably slower. DeepSeek-R1, as a reasoning model, is designed to map its step-by-step reasoning process to the final answer, which may explain its high latency in exchange for strong AR and CP. However, the latency gap between DeepSeek-V3 and GPT-4.1 remains large. DeepSeek-V3 has 671B parameters totally and only 37B are “activated” for handling each token, implying it may still require caching a large number of inactive weights. While DeepSeek models adopt a number of refined architecture strategies such as MLA and MTP and offer excellent FQ for UX, further optimization of its structure remains necessary to reduce possible latency [52, 38].

Chapter 5

Research Evaluation

This chapter evaluates the project based on several factors. Section 5.1 investigates the effect of different GPU devices, chunk sizes and overlaps on RAG engine construction time. Section 5.2 further examines how different overlap sizes influence the metric of parameter recommendation tasks. Section 5.3 provides a suggestion for optimizing the table data structure. Section 5.4 compares part of this project's results with the related work. Section 5.5 discusses validity threats and Section 5.6 summarizes key research findings. Section 5.7 presents reflections on the study.

5.1 RAG Engine Construction Time under Different Conditions

5.1.1 Different GPU Devices

As introduced in Section 2.4.1, the construction of the RAG system consists of several stages. In this project, the time required for each stage under different GPU configurations [5] was systematically recorded, with **a chunk size of 512 and an overlap of 50**, without the expert engine for table data. Detailed results are presented in Table 5.1. Explanations for each column are as follows:

- "Number of Document Pages": the number of literature PDF pages loaded as source documents.
- "Number of Chunks": the number of chunks (segments) into which the PDF pages are divided.
- "Read Time": the time required for the system to load the original document pages.
- "Parse Time": the time spent splitting document pages into chunks.
- "Create Time": the time spent converting chunks into vectors using the embedding model and creating index mappings.
- "Save Time": the time taken to save converted vectors and index mappings locally.

Table 5.1: Detailed Time Statistics for RAG Engine Construction Across Various GPU Devices *(Using Chunk Size 512 and Overlap 50)*

GPU Types	Index Name	Total Build Time (Second)	Number of Document Pages	Number of Chunks	Read Time (Second)	Parse Time (Second)	Create Time (Second)	Save Time (Second)
RTX-A6000	Global Engine Index	294.65	2505	7607	90.75	9.31	166.16	28.43
	Expert Engine - FDM Process Parameters	227.62	1482	4135	119.20	4.12	88.86	15.43
	Expert Engine - General Knowledge of PLA and Fillers	165.16	981	3307	76.79	3.39	72.59	12.38
	Expert Engine - General Knowledge of PLA Itself	207.36	1320	3963	103.27	3.96	85.32	14.78
RTX-4090	Global Engine Index	218.84	2505	7607	95.87	9.32	82.16	31.49
	Expert Engine - FDM Process Parameters	190.21	1482	4135	125.59	3.64	44.11	16.86
	Expert Engine - General Knowledge of PLA and Fillers	133.82	981	3307	81.56	2.96	35.75	13.53
	Expert Engine - General Knowledge of PLA Itself	165.87	1320	3963	103.99	3.50	42.20	16.16
RTX-3090	Global Engine Index	314.06	2505	7607	100.41	10.81	178.44	24.40
	Expert Engine - FDM Process Parameters	253.81	1482	4135	140.62	5.00	94.94	13.24
	Expert Engine -General Knowledge of PLA and Fillers	177.57	981	3307	85.70	3.87	77.47	10.52
	Expert Engine - General Knowledge of PLA Itself	229.87	1320	3963	120.57	4.99	91.54	12.75
Tesla-V100	Global Engine Index	388.95	2505	7607	77.43	8.95	280.36	22.21
	Expert Engine - FDM Process Parameters	282.61	1482	4135	117.25	3.20	150.05	12.10
	Expert Engine - General Knowledge of PLA and Fillers	205.61	981	3307	71.73	2.57	121.62	9.67
	Expert Engine - General Knowledge of PLA Itself	253.17	1320	3963	94.67	3.07	143.80	11.61
Apple-MPS	Global Engine Index	1098.31	2505	7627	42.76	3.60	1038.59	13.36
	Expert Engine - FDM Process Parameters	626.77	1482	4144	58.78	1.74	558.88	7.35
	Expert Engine - General Knowledge of PLA and Fillers	497.61	981	3310	37.77	1.41	452.58	5.85
	Expert Engine - General Knowledge of PLA Itself	592.41	1320	3973	48.79	1.71	534.92	7.00

The local MPS achieves shorter Read Time, Parse Time and Save Time compared to the GPU servers. Recorded terminal logs from GPU servers indicate many system-level operations are less streamlined, often accompanied by extensive hardware-related messages. On the local machine, these processes are

more straightforward. Nonetheless, for floating-point operations in embedding vector generation, GPUs provide a clear advantage in speed. And there are some interesting observations: Tesla V100 has the shortest Read Time compared to RTX GPUs. But its performance in floating-point computation, as reflected by the Create Time, is weaker. Besides, with the same Number of Document Pages, the local MPS generates more chunks than the GPU servers.

5.1.2 Different Chunk Sizes

The project evaluated the impact of different chunk sizes (512, 1024 and 2048) on RAG engine construction time with a fixed overlap of 50 in the RTX-A6000 environment, as shown in Table 5.2. As the chunk size increases, document pages are divided into fewer chunks, and the processing time decreases. This results in each chunk containing more content.

Table 5.2: Time Required for RAG Engine Construction Across Different Chunk Sizes
(Overlap Fixed at 50)

Chunk Sizes	Index Name	Total Build Time (Second)	Number of Document Pages	Number of Chunks
512	Global Engine Index	294.65	2505	7607
1024	Global Engine Index	202.77	2505	4141
2048	Global Engine Index	168.25	2505	3783

5.1.3 Different Overlap Sizes

The project also evaluated the impact of different overlap sizes (50, 100 and 200) on RAG engine construction time with a fixed chunk size of 512 in the RTX-A6000 environment, as shown in Table 5.3. Because each chunk overlaps with the previous one, increasing the overlap size reduces the stride between chunks. As a result, a larger overlap results in more chunks from the same document, thereby extending the overall building time.

Table 5.3: Time Required for RAG Engine Construction Across Different Overlap Sizes
(Chunk Size Fixed at 512)

Overlap Sizes	Index Name	Total Build Time (Second)	Number of Document Pages	Number of Chunks
50	Global Engine Index	294.65	2505	7607
100	Global Engine Index	316.88	2505	8123
200	Global Engine Index	346.38	2505	9666

5.2 Effects of RAG Configurations on System Performance

The challenges of RAG applications mainly come from interactions between chunking approaches, the embedding model and other modules. As shown in Tables 5.2 and 5.3, different chunk and overlap sizes affect "retrieval granularity", which may influence both storage requirements and construction efficiency of the underlying retrieval system [22, 15].

5.2.1 For Initially Unanswerable 85 Parameter Recommendation Questions Using Table Data

This project further examined how the overlap size impacted system behaviors [22]. Using GPT-4o as the generator and table data listed in Appendix A.3, different RAG engines were built with a fixed chunk size of 512 and different overlaps (50, 100 and 200) to answer 85 FDM parameter recommendation questions that could not be directly answered by textual literature data. Evaluation was then conducted using metrics defined in Section 3.4.3. Results are listed in Table 5.4.

As shown in Table 5.4, reducing the overlap leads to a decline in metrics. The drop in hit rate indicates that the system starts to reject some questions, even when relevant information exists in table data. When the overlap was set to 100, there is a notable decrease in precision, suggesting the system makes more incorrect recommendations. Manual inspection reveals that some errors are simply due to the model returning incorrect values, while others are caused by the model providing excessive or hallucinatory information [15]. The recall metric exhibits the same downward trend. Questions incorrectly answered with overlaps of 50 and 100 are listed in Appendix A.4 and Appendix A.5, respectively. A possible reason for the decline is that a smaller overlap may fail to capture all relevant information for a query within a single chunk, causing essential content to be split across chunks and missed during retrieval.

Table 5.4: Evaluation Results for 85 Parameter Recommendation Questions under Different Overlap Sizes *(Chunk Size Fixed at 512)*

Overlap Sizes	Data Coverage Rate	Hit Rate	Precision	Recall
200	0.61	1.00	1.00	1.00
100	0.61	0.98	0.88	0.87
50	0.61	0.98	0.98	0.96

5.2.2 For All Parameter Recommendation Questions Using Textual Literature Data

The project also constructed an additional expert engine for "FDM Process Parameters" using GPT-4o with a chunk size of 512 and an overlap of 200. In this case, all 100 parameter recommendation questions were re-answered. The number of answered questions are shown in Table 5.5. The "Baseline" in Table refers to the initial experiment, described in Section 4.1, where GPT-4o was able to answer only 15 out of 100 parameter recommendation questions using the "FDM Process Parameter" expert engine with a chunk size of 512 and an overlap of 50.

Table 5.5 shows that with a larger overlap, the system is able to answer more parameter recommendation questions based on retrieved document segments. As mentioned in Section 5.1.3, increasing the overlap provides "finer-grained retrieval units", which results in more literature segments and enables the system to answer more parameter recommendation questions. However, for the "FDM Process Parameter" expert engine with over 100 documents, building such engines with different configurations is time-consuming and leads to higher resource consumption [15]. Also, this approach did not yield ideal performance for parameter recommendations, even when the overlap was increased to 200.

Table 5.5: Number of All 100 Parameter Recommendation Questions Answered under Different Experiment Settings

Experiment Settings	Chunk Size	Overlap Size	Number of Answered Questions
Baseline (Original Configurations)	512	50	15
Evaluation Experiment	512	200	19

5.2.3 Insights from Comparative Evaluation

The creation time of expert engines for literature data and tabular data was recorded. As shown in Table 5.6, using structured table data in HTML format requires only about two seconds to create an engine, enables the system to answer more parameter recommendation questions correctly, and allows for flexible hyperparameter adjustments during testing. Conversely, unstructured textual literature data leads to longer engine creation times and worse recommendation performance. Therefore, when designing the chunking strategy, the type of indexed data should be taken into account. For example, unstructured textual data is more suitable for text-based QA, while tasks such as parameter recommendations, requiring accurate numerical extraction [58], may benefit more from using structured data like tables.

Additionally, different data types likely require different levels of granularity. For structured data, which typically contain less content than literature data, it is feasible to use a larger overlap to ensure that each chunk contains more complete information [44]. In contrast, for unstructured data such as literature, setting a high overlap may not be practical due to the significantly higher retrieval cost, as shown in Table 5.1 and Table 5.6.

As a result, if users have diverse Functional Requirements (FRs) such as open-ended academic QA and parameter recommendations, the final product may need to adopt different chunking configurations and manage various types of data files [15, 44]. This demonstrates the requirement for code-driven RAG database management [63], whether by dividing files into different folders based on content or creating different engines using appropriate chunking strategies.

Besides, this evaluation verifies a phenomenon: mismatches in RAG systems mainly arise from shortcomings in the retrieval module, rather than from LLMs with prompt engineering [44].

Table 5.6: Time Required for Different Expert Engine Construction
(Chunk Size Fixed at 512 and Overlap Fixed at 200 on local MPS)

Index Name	Number of Document Pages	Number of Chunks	Total Build Time (Second)
Expert Engine - FDM Process Parameters	1482	5226	781.87
Expert Engine - Enhanced Table Data	1	6	1.17

5.3 Table Data Structure Optimization

During parameter recommendation experiments using the table data, another noteworthy issue was observed. In the initial table, multiple recommended values for a FDM parameter were separated by commas. However, the LLM sometimes generated range-like responses, such as "from...to..." when answering. A practical solution is to separate multiple recommended values using the word "or" in the table, as shown in Appendix A.3. This effectively resolved the issue by improving the table data structure [15], as demonstrated in Table 5.7.

Table 5.7: Effects of Table Data Structure Optimization on Response Styles

Parameter Recommendation Questions	Ambiguous Responses	Improved Responses
What is the appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF)?	The appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is 210°C to 250°C. This information is sourced from the \"Enhanced Table Data\" document.	The appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is either 210°C or 250°C, as specified in the document \"Enhanced_Table_Data.html\" from the 'Enhanced Table Data' folder.
What is the appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF)?	The appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is 3.0 mm/s to 25 mm/s. This information is sourced from the \"Enhanced Table Data\" document.	The appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is either 3.0 mm/s or 25 mm/s, as specified in the document \"Enhanced_Table_Data.html\" from the 'Enhanced Table Data' folder.
What is the appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF)?	The appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is 0.2 mm to 0.5 mm. This information is sourced from the \"Enhanced Table Data\" document.	The appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is either 0.2 mm or 0.5 mm, as specified in the document \"Enhanced_Table_Data.html\" from the 'Enhanced Table Data' folder.
What is the appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF)?	The appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is 60°C to 90°C. This information is sourced from the \"Enhanced Table Data\" document.	The appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with short carbon fibre (SCF) is 60°C, 75°C, or 90°C, as specified in the document \"Enhanced_Table_Data.html\" from the 'Enhanced Table Data' folder.

5.4 Comparison with Related Work

[58] investigated different RAG index structures and evaluated the performance of GPT-4o and Deepseek-V2 in responding to SM queries, assessing both CP and Latency metrics. Although they did not release a complete set of data or questions, nor did they specify an explicit target domain, they did provide several example questions related to abstract manufacturing explanations, which could be viewed as open-ended queries.

In their work, for the vector-based RAG method, which is consistent with this project, GPT-4o outperformed Deepseek-V2 in terms of CP, while our findings showed DeepSeek-V3 surpassed GPT-4o. This suggests that, with advancements such as MTP and the knowledge distillation from DeepSeek-R1 reasoning abilities [38, 21], DeepSeek-V3 has likely overtaken GPT-4o in handling text-only applications. In addition, their experiments reported GPT-4o had the slowest inference speed compared to DeepSeek-V2, whereas our experiments showed GPT-4o was much faster than both DeepSeek-V3 and R1. This further implies that for open-ended QA, although DeepSeek models can generate highly relevant and well-explained answers from retrieved content, their architectures still lead to slower inference speed, which may limit their deployments in scenarios requiring rapid responses.

5.5 Validity Threats

5.5.1 Construct Validity

1. For open-ended QA tasks, DeepSeek-V3 was used as an evaluator to assess generated answers from different LLMs across widely recognized dimensions such as AR and CP [58, 15]. To reduce the impact of randomness caused by API calls [26], each evaluation was repeated three times per dimension, and the average score was used for analysis. Moreover, prompts of LLMs for both answer generation and evaluation were refined based on published best practices [44, 63, 58], to ensure the validity of results.

2. For parameter recommendation tasks utilizing table data, task-specific metrics such as hit rate and precision [15] were defined and employed to enable more rigorous comparison and analysis of the system performance.

3. Since all LLMs were accessed via online APIs, latency measurements may be imprecise. To address this threat, three questions were selected for LLMs, and the response time was recorded three times per question. The average response time of answering three questions was then used as the representative latency for analysis.

5.5.2 Internal Validity

1. The large volume of literature data were not explicitly verified by manual inspection. However, they were collected through backward snowballing from review articles on FDM using PLA, sourced from

publishers such as ScienceDirect and Emerald Insight, and were indexed by Google Scholar, which substantially reduces the associated risk.

2. During parameter recommendation tasks, after optimizing the chunking strategy, some evaluation metrics reached a perfect score of 1.0. This is because, as defined in Section 3.4.3, these metrics were primarily calculated based on questions for which answers are available in the table data. If larger table datasets and a broader range of questions are introduced, these metrics may not remain at such high levels. To facilitate critical analysis, the **data coverage rate** was defined as an additional metric for a more balanced assessment.

3. Due to time and scale constraints, only GPT-4o was used for parameter recommendations. There is a possibility that other LLMs, when provided with the same retrieved knowledge, would demonstrate different behaviors in response to the same questions compared to GPT-4o. However, this is less likely to substantially affect the conclusions regarding the impact of different chunk configurations on system performance.

4. Although the chat function was implemented in this project, as shown in Figure 3.2, both parameter recommendations and open-ended QA modules were ideally intended to operate in a querying mode [5], where each session would consist of a single question and answer. However, due to time constraints, three to five questions were submitted within a single conversation during evaluation to collect responses efficiently. This approach may have caused the model to retain prior conversation history, potentially affecting some experimental outcomes. Nevertheless, given the carefully designed prompt templates, this instability is considered acceptable.

5. The APIs of LLMs used in this project are publicly accessible. However, GPT-4o and GPT-4.1 themselves are closed-source, and their exact architectures remain unknown. Therefore, when comparing them with DeepSeek models, certain assumptions had to be made based solely on their observed performance.

6. In assessing the 45 open-ended queries with DeepSeek-V3, 15 parameter recommendation questions were included because their answers were all retrieved from textual literature data. These recommended values in responses were not manually verified, so, as in Section 5.2.1, errors or hallucinations may have arisen from chunk segmentation or overlap settings. The remaining 30 open-ended questions also lacked ground truth, as their design encouraged broader reference coverage rather than limiting answers to the initial reviews (see Section 3.1.4). Furthermore, none of the results underwent expert review. These factors may have affected the internal validity of this project.

5.5.3 External Validity

1. The procedures for collecting both literature and table data are open and transparent. With the exception of the literature data and certain engines, which were not uploaded due to file size limitations, all related files have been made available on GitHub[1]. This enables future researchers to conduct further critical analysis and comparison.

2. This project implements a chat system with the frontend and backend architecture, and integrates a code-based file management mechanism that can automatically create RAG chat engines for different folders. This design also provides a solid foundation for future research, such as developing new engines and can be applied not only in AM but also in other practical domains.

3. Although literature on chat systems based on RAG and LLMs in AM remains relatively limited, this project expanded the scope of the literature review by conducting both forward and backward snowballing from an existing querying system focused on metal AM [5]. In addition, this report provides a clear and detailed overview of the development and fundamental principles of LLMs, as well as an in-depth introduction to open-source DeepSeek models. As such, this report can serve as a valuable starting point for backward snowballing, whether for engineers with limited AI background or for AI researchers who are interested in this domain but have little engineering experience.

[1] https://github.com/ZihaoChen189/AM_Chat_System_based_on_RAG_and_LLMs

5.5.4 Reproducibility

LLM API suppliers have full control regarding LLMs and may implement unannounced modifications [26]. In this project, all LLMs were accessed via APIs, and despite fixing the sampling temperature at zero, LLMs especially OpenAI GPT products [7], frequently generate variable responses [26]. Additionally, as shown in Table 5.1, different devices or systems may yield different chunk counts. Therefore, the reproducibility of this project is subject to certain threats. To mitigate this, specific model versions were referenced in the code such as `"model":"gpt-4.1-2025-04-14"`, and random seeds were set consistently across different Python libraries.

5.6 Summary of Key Research Findings

1. When deploying RAG in scenarios requiring a literature database, selecting a few key review articles and conducting targeted backward snowballing is an effective way to collect relevant literature, especially when resources are constrained and manual verification is difficult.

2. For applications where FRs span multiple tasks such as open-ended questions and parameter recommendations, a code-based data management approach, with dedicated engines for each task, is advisable. This approach, as used in this project, enhances the adaptability of RAG systems.

3. For tasks requiring precise numerical recommendations by RAG, simply increasing the amount of textual literature database or using finer-grained chunking is not often optimal. Instead, collecting relevant information and manually creating structured data like tables may be more effective [28, 15, 22].

4. For lightweight structured table data, a high degree of overlap can help maximize the information contained within each chunk for retrieval. However, for unstructured textual literature, it is important to offset retrieval effectiveness against the increased computational cost associated with finer-grained chunking [32].

5. The outputs of various LLMs demonstrate notable stylistic and functional differences in practical applications. First, DeepSeek-V3 shows improved text processing over V2, partly due to architectural enhancements such as MTP. However, compared to OpenAI products such as GPT-4o and GPT-4.1, it still exhibits significantly higher latency, a concern that also applies to DeepSeek-R1. Second, without output constraints, DeepSeek models, especially V3, often use Markdown tags and ask follow-up questions, while GPT-4o tends to give direct and brief answers with little Markdown. Third, in English language contexts, responses from GPT models tend to be more coherent, with more logical connections between sentences, compared with DeepSeek models. Moreover, both DeepSeek models and GPT-4.1 exhibit strong information integration. With appropriate prompts, GPT-4o demonstrates effective negative rejection. Finally, if output length is not restricted in the prompt, DeepSeek-R1 tends to use many tokens for Markdown tags and elaborates more for each point, increasing the risk of truncation and affecting UX.

5.7 Reflection

Another possible reason for the suboptimal performance of textual literature data in FDM parameter recommendations is inconsistent parameter naming. For example, as noted in Section 3.1.4, Review3 [29] used both “print speed” and “printing speed”, while [18] referred to “nozzle temperature” as “print temperature”. These inconsistencies may cause “concept confusion” [8]. Due to resource constraints, this project did not fully standardize all parameter names, which may have led to missing relevant text chunks during retrieval. This highlights the need for a unified naming framework for FDM parameters in future research, whether for pure PLA or PLA composites. In addition, this project did not further investigate whether the long latency observed for DeepSeek models was related to IP-based factors. Finally, DeepSeek-V3 served as both evaluator and evaluated model in this study. This may introduce some bias, but since DeepSeek-V3 was not the top performer on all metrics, the overall findings are unlikely to be affected.

Chapter 6

Conclusion

This chapter provides the conclusion, with Section 6.1 summarizing achievements and Section 6.2 offering suggestions for future work.

6.1 Achievements and Contributions

1. This project implements a chat system based on RAG and LLMs, focusing on the use of PLA in FDM within AM. It also serves as a practical case of using RAG to mitigate LLM hallucinations [54] through precise citation of published literature [5], and offers deployment experiences [15, 44]. Furthermore, it reduces the effort spent by researchers and non-expert users to read technical documents [5, 31].
2. A total of 30 open-ended academic questions concerning FDM and PLA are developed, as well as 100 FDM parameter recommendation questions for PLA and its composites. These resources can be contributed to FDM-Bench [13], further enriching this dataset.
3. With a code-driven database management approach, the chat system can address open-ended queries from textual literature and offer precise parameter recommendations [14] from structured HTML tabular data [59, 15, 58]. Employing zero-shot prompts [33], it also exhibits strong abilities in information integration and negative rejection [8].
4. Using recognized assessment dimensions and an LLM-based evaluation method [58, 8], the differences in RAG outputs with various LLMs have been investigated [15, 6], particularly for open-ended academic queries. In addition, defined metrics are used to evaluate the effect of different chunk overlap sizes on GPT-4o's performance in FDM parameter recommendation tasks.
5. This project provides a comprehensive literature review in AM, RAG and LLMs. For the data science community, this report can serve as a valuable starting point for snowballing and quickly accumulating literature and knowledge related to AM, RAG and LLMs.

6.2 Suggestions for Future Work

1. As discussed in Section 5.5.2, this project lacks expert review [6]. Future work could address this gap and employ metrics such as faithfulness and Exact Match (EM) [58, 59, 15, 22] for more comprehensive evaluation of hallucination mitigation by RAG with different LLMs in practice [8].
2. A multimodal embedding model could be used to integrate knowledge derived from figures and tables in literature, further enriching the RAG database [5, 63, 13, 59, 28, 15, 22, 14, 35].
3. Additional expert engines, such as FDM-related G-code [13, 43], could be developed in parallel within the current framework. The architecture for handling various FRs in this project may also inform similar engineering applications.
4. This project did not achieve separate chunk and hyperparameter configurations for different expert engine construction, which could be further refined in future work. In addition, alternative chunk methods beyond the fixed chunk length approach could be further explored. The saved conversation history [5] may also be further trained for "local user preferences" [63].

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Assem Al-Hawari, Hassan Najadat, and Raed Shatnawi. Classification of application reviews into software maintenance tasks using data mining techniques. *Software Quality Journal*, 29(3):667–703, 2021.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Achuth Chandrasekhar, Jonathan Chan, Francis Ogoke, Olabode Ajenifujah, and Amir Barati Farimani. Amgpt: a large language model for contextual querying in additive manufacturing. *Additive Manufacturing Letters*, 11:100232, 2024.

[6] Achuth Chandrasekhar, Omid Barati Farimani, Olabode T Ajenifujah, Janghoon Ock, and Amir Barati Farimani. Nanogpt: A query-driven large language model retrieval-augmented generation system for nanotechnology research. *arXiv preprint arXiv:2502.20541*, 2025.

[7] Hailin Chen, Fangkai Jiao, Xingxuan Li, Chengwei Qin, Mathieu Ravaut, Ruochen Zhao, Caiming Xiong, and Shafiq Joty. Chatgpt's one-year anniversary: are open-source large language models catching up? *arXiv preprint arXiv:2311.16989*, 2023.

[8] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762, 2024.

[9] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

[10] KS Derekar. A review of wire arc additive manufacturing and advances in wire arc additive manufacturing of aluminium. *Materials science and technology*, 34(8):895–916, 2018.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[13] Ahmadreza Eslaminia, Adrian Jackson, Beitong Tian, Avi Stern, Hallie Gordon, Rajiv Malhotra, Klara Nahrstedt, and Chenhui Shao. Fdm-bench: A comprehensive benchmark for evaluating large language models in additive manufacturing tasks. *arXiv preprint arXiv:2412.09819*, 2024.

[14] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501, 2024.

[15] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

[16] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[17] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[18] Mohamed Hamoud, Abou Bakr Elshalakany, Mohammed Gamil, and Hussien Mohamed. Investigating the influence of 3d printing parameters on the mechanical characteristics of fdm fabricated (pla/cu) composite material. *The International Journal of Advanced Manufacturing Technology*, 134(7):3769–3785, 2024.

[19] Simon Harper. Ux from 30,000ft.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[22] Yizheng Huang and Jimmy Huang. A survey on retrieval-augmented text generation for large language models. *arXiv preprint arXiv:2404.10981*, 2024.

[23] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[24] Aniket Jadhav and Vijay S Jadhav. A review on 3d printing: An additive manufacturing technology. *Materials Today: Proceedings*, 62:2094–2099, 2022.

[25] Yayati Jadhav, Peter Pak, and Amir Barati Farimani. Llm-3d print: large language models to monitor and control 3d printing. *arXiv preprint arXiv:2408.14307*, 2024.

[26] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023.

[27] Katikapalli Subramanyam Kalyan. A survey of gpt-3 family large language models including chatgpt and gpt-4. *Natural Language Processing Journal*, 6:100048, 2024.

[28] Balduin Katzer, Steffen Klinder, and Katrin Schulz. Towards an automated workflow in materials science for combining multi-modal simulation and experimental information using data mining and large language models. *Materials Today Communications*, 45:112186, 2025.

[29] Abdullah Burak Keşkekçi, Merdan Özkahraman, and Hilmi Cenk Bayrakçı. A review on the impact of polylactic acid (pla) material on products manufactured using fused deposition modeling (fdm) additive manufacturing method. *Gazi Mühendislik Bilimleri Dergisi*, 9(4):158–173, 2023.

[30] Kiarash Naghavi Khanghah, Zhiling Chen, Lela Romeo, Qian Yang, Rajiv Malhotra, Farhad Imani, and Hongyi Xu. Multimodal rag-driven anomaly detection and classification in laser powder bed fusion using large language models. *arXiv preprint arXiv:2505.13828*, 2025.

[31] Kiarash Naghavi Khanghah, Anandkumar Patel, Rajiv Malhotra, and Hongyi Xu. Large language models for extrapolative modeling of manufacturing processes. *arXiv preprint arXiv:2502.12185*, 2025.

[32] Grigorii Khvatskii, Yong Suk Lee, Corey Angst, Maria Gibbs, Robert Landers, and Nitesh V Chawla. Do multimodal large language models understand welding? *Information Fusion*, 120:103121, 2025.

[33] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[34] Vladimír Kunc and Jiří Kléma. Three decades of activations: A comprehensive survey of 400 activation functions for neural networks. *arXiv preprint arXiv:2402.09092*, 2024.

[35] Hugon Lee, Hyeonbin Moon, Junhyeong Lee, and Seunghwa RYu. Toward knowledge-guided ai for inverse design in manufacturing: A perspective on domain, physics, and human-ai synergy. *arXiv preprint arXiv:2506.00056*, 2025.

[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[37] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[38] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[39] Zengguang Liu, Yanqing Wang, Beicheng Wu, Chunzhi Cui, Yu Guo, and Cheng Yan. A critical review of fused deposition modeling 3d printing technology in manufacturing polylactic acid parts. *The International Journal of Advanced Manufacturing Technology*, 102(9):2877–2889, 2019.

[40] A Neelima and Shashi Mehrotra. A comprehensive review on word embedding techniques. In *2023 International Conference on Intelligent Systems for Communication, IoT and Security (ICISCoIS)*, pages 538–543. IEEE, 2023.

[41] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[42] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[43] Zongrui Pei, Junqi Yin, and Jiaxin Zhang. Language models for materials discovery and sustainability: Progress, challenges, and opportunities. *Progress in Materials Science*, page 101495, 2025.

[44] Sonal Prabhune and Donald J Berndt. Deploying large language models with retrieval augmented generation. *arXiv preprint arXiv:2411.11895*, 2024.

[45] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[46] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[47] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[48] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[49] L Sandanamsamy, WSW Harun, I Ishak, FRM Romlay, K Kadirgama, D Ramasamy, SRA Idris, and F Tsumori. A comprehensive review on fused deposition modelling of polylactic acid. *Progress in Additive Manufacturing*, 8(5):775–799, 2023.

[50] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[51] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[52] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[53] Dachuan Shi, Jianzhang Li, Olga Meyer, and Thomas Bauernhansl. Enhancing retrieval-augmented generation for interoperable industrial knowledge representation and inference toward cognitive digital twins. *Computers in Industry*, 171:104330, 2025.

[54] SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*, 6, 2024.

[55] François Torregrossa, Robin Allesiardo, Vincent Claveau, Nihel Kooli, and Guillaume Gravier. A survey on training and evaluation of word embeddings. *International journal of data science and analytics*, 11(2):85–103, 2021.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[57] Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, et al. Freshllms: Refreshing large language models with search engine augmentation. *arXiv preprint arXiv:2310.03214*, 2023.

[58] Yuwei Wan, Zheyuan Chen, Ying Liu, Chong Chen, and Michael Packianather. Empowering llms by hybrid retrieval-augmented generation for domain-centric q&a in smart manufacturing. *Advanced Engineering Informatics*, 65:103212, 2025.

[59] Zhiqi Wang, Zhongcun Liu, Weizhen Lu, and Lu Jia. Improving knowledge management in building engineering with hybrid retrieval-augmented generation framework. *Journal of Building Engineering*, 103:112189, 2025.

[60] Zhongju Wang, Long Wang, Chao Huang, Shutong Sun, and Xiong Luo. Bert-based chinese text classification for emergency management with a novel loss function. *Applied Intelligence*, 53(9):10417–10428, 2023.

[61] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[63] Chengke Wu, Wenjun Ding, Qisen Jin, Junjie Jiang, Rui Jiang, Qinge Xiao, Longhui Liao, and Xiao Li. Retrieval augmented generation-driven information retrieval and question answering in construction management. *Advanced Engineering Informatics*, 65:103158, 2025.

[64] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 641–649, 2024.

Appendix A

Appendix

A.1 Visualisation of Backward Snowballing Starting from AMGPT

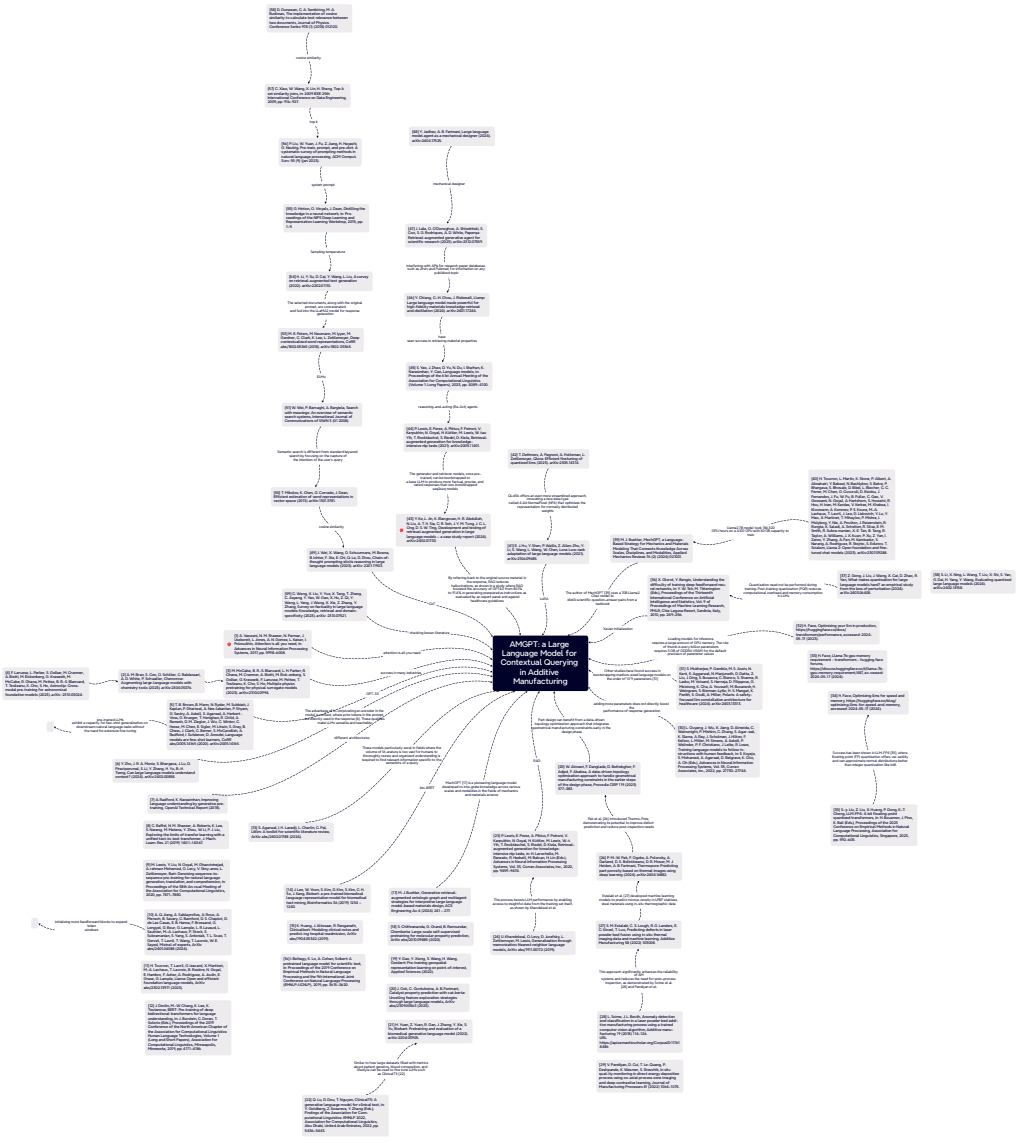


Figure A.1: Visualisation of Backward Snowballing Process Starting from AMGPT

A.2 Traditional and Static Embeddings

Traditional embeddings are primarily based on statistical features such as Term Frequency-Inverse Document Frequency (TF-IDF) [44] and co-occurrence information [40]. TF-IDF considers both the frequency of a term in a single document and the number of documents in which it appears across the entire corpus, thereby determining the significance of the term within the text [2]. The co-occurrence embeddings aim to identify words that frequently appear together, thereby uncovering potential semantic relations [40].

Static embeddings assign a distinct numerical representation to each token in the vocabulary during training on the dataset. Once determined, the representation stays unchanged regardless of the sentence in which it appears during downstream applications [55]. For example, Word2Vec [40, 44] can derive static embeddings in two ways: one is by inferring surrounding terms provided a central term, known as the Skip-Gram [43]; the other is by inferring the central term based on surrounding terms, known as the Continuous Bag-of-Words (CBOW). Compared to Skip-Gram, CBOW may capture more grammar features rather than conceptual ones [55].

A.3 Table Data for FDM Printing Parameters of PLA Reinforced with Various Fillers

Table A.1: Table Data for FDM Printing Parameters of PLA Reinforced with Various Fillers

HTML Table Data for Fused Deposition Modeling (FDM) Printing Parameters of Polylactic Acid (PLA) Reinforced with Various types of Fillers

PLA–based Composite	Layer Height (mm)	Nozzle Diameter (mm)	Nozzle Temperature (°C)	Platform (Bed) Temperature (°C)	Infill Density (%)	Printing Speed	Raster Angle (°)	Infill Pattern
PLA + Copper (Cu)	0.8	0.4	220	60	–	30 mm/sec	70	–
PLA + Wood	0.3	0.4	200	65	100	80 mm/min	0 or 90 or 45 or –45	Linear
PLA + hydroxypropyl methylcellulose (HPMC)	0.2	–	200	40	100	50 mm/s	0	Parallel
PLA + microcrystalline cellulose (MCC)	0.38	–	200	60	100	60 mm/s	45 or –45	–
PLA + carbonised cellulose nanofibers (CCN)	0.2	0.4	220	70	100	2400 mm/min	45 or –45	Rectilinear
PLA + cellulose nanofibers (CNF)	0.2	0.6	210	60	100	1800 mm/min	45 or –45	Rectilinear
PLA + cellulose nanocrystal (CNC)	0.2	0.35	230	60	100	–	–	–
PLA + Cork Powder	0.4	0.8	230	60	100	30 mm/s	–	–
PLA + Hydroxyapatite (HA)	0.4	0.4	210	50	–	35 or 50 mm/s	–	–
PLA + Akermanite	0.4	0.4	210	–	–	10 mm/s	–	–
PLA + Carbon	0.1 or 0.2 or 0.3	–	–	–	65 or 75 or 85	60 or 80 or 100 mm/s	–	–
PLA + Carbon Black	0.2	0.4	230	60	100	30 mm/s	0 or 45 or – 45 or 90	–
PLA + Carbon Fibre (CF)	0.1 or 0.2 or 0.3	0.4	205 or 215 or 225	–	40 or 60 or 80	60 or 80 or 100 mm/s	–	–
PLA + continuous carbon fibre (CCF)	0.2 or 0.5	0.4 or 1.5	210 or 250	60 or 75 or 90	100	3.0 or 25 mm/s	–	Rectilinear
PLA + short carbon fibre (SCF)	0.2 or 0.5	0.4 or 1.5	210 or 250	60 or 75 or 90	100	3.0 or 25 mm/s	–	Rectilinear

A.4 Base Prompt

You are an intelligent conversational assistant for additive manufacturing based on Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). Answer user questions strictly based on the retrieved document content and cite the specific document source in your answers. If there is no relevant information in the retrieved document content, explicitly state: I do not know. Do not generate answers that are not supported by the retrieved document content. If asking the user a clarifying question would be helpful, please do so.

A.5 Prompt for Expert Engines

You are an intelligent conversational assistant for additive manufacturing based on Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). You specialize in answering user questions strictly based on the content of documents retrieved from the 'folder_name' folder. In your answers, always cite the specific document source. If there is no relevant information in the retrieved document content, explicitly state: I do not know. In this case, do not cite any document source. Do not generate any answers that are not supported by the retrieved document content. If asking the user a clarifying question would be helpful, please do so.

A.6 Prompt for LLM-Based Evaluation (Old Version)

```
"""Below is a question and a student's answer based on literature about
Additive Manufacturing, specifically regarding
Fused Deposition Modeling (FDM) and Polylactic Acid (PLA).
The question and the referenced literature may also include
content related to PLA composites.

**Question:**
{question}
**Student Answer:**
{candidate}

Evaluate the answer on the following dimensions (score each from 1 to 10,
decimals allowed, 10 = best):

1. Answer_Relevance: The degree to which the content of the
student's answer is relevant to the question.
2. Context_Precision: The proportion of information in the student's answer
that is relevant to the question, relative to all information
provided in the answer.
3. Completeness: Whether the student's answer is complete, for example,
whether there are any incomplete sentences.
4. Coherency: Whether the meaning of the student's answer is smooth and clear,
for example, how well the sentences are connected to each other.
5. Formatting_Quality: Evaluate the use of Markdown.
Consider whether Markdown syntax is present and whether the
formatting meaningfully improves readability and organization.

Also provide an Explanation: Give the rationale for each score or
identify potential issues, covering all dimensions.

Output only valid JSON in the following format, with no additional text
outside the JSON.
The values in this example are placeholders and should
not be taken as the reference.

{{
  "Answer_Relevance": 5.0,
  "Context_Precision": 5.0,
  "Completeness": 5.0,
  "Coherency": 5.0,
  "Formatting_Quality": 5.0,
  "Explanation": "Scoring rationale or potential issues if exist."
}}

Note:
- Evaluate objectively and rigorously, and do not be lenient in scoring.
"""
```

A.7 Prompt for LLM-Based Evaluation (New Version)

```
"""Below is a question and a student's answer based on literature about
Additive Manufacturing, specifically regarding
Fused Deposition Modeling (FDM) and Polylactic Acid (PLA).
The question and the referenced literature may also include
content related to PLA composites.

**Question:**
{question}
**Student Answer:**
{candidate}

Evaluate the answer on the following dimensions (score each from 1 to 10,
decimals allowed, 10 = best):

1. Answer_Relevance: The degree to which the content of the
student's answer is relevant to the question.
2. Context_Precision: The proportion of information in the student's answer
that is relevant to the question, relative to all information
provided in the answer.
3. Completeness: Whether the student's answer is complete, for example,
whether there are any incomplete sentences.
4. Coherency: Whether the meaning of the student's answer is smooth and clear,
for example, how well the sentences are connected to each other.
5. Formatting_Quality: Evaluate the use of Markdown.
Consider whether Markdown syntax is present and whether the
formatting meaningfully improves readability and organization.

Also provide an Explanation: Give the rationale for each score or
identify potential issues, covering all dimensions.

Output only valid JSON in the following format, with no additional text
outside the JSON.
The values in this example are placeholders and should
not be taken as the reference.

{{
  "Answer_Relevance": 5.0,
  "Context_Precision": 5.0,
  "Completeness": 5.0,
  "Coherency": 5.0,
  "Formatting_Quality": 5.0,
  "Explanation": "Scoring rationale or potential issues if exist."
}}

Note:
- Evaluate objectively and rigorously, and do not be lenient in scoring.
- Score EACH dimension independently. Do not blend dimensions or compensate
a low score in one dimension with a high score in another. Do not compute or
imply an overall/average score.
- Completeness: if any sentence or paragraph is clearly truncated,
apply an additional penalty on this dimension,
as truncation indicates missing content.
- Formatting_Quality: if the answer, even when short, could be made clearer
with basic Markdown but none is used, deduct points; also deduct for incorrect
or harmful formatting.
"""
```

A.8 Detailed Scores from Three Evaluation Trials for Each LLM (AR, CP, Completeness, Coherency, FQ)

Table A.2: Detailed Scores from Three Evaluation Trials for Each LLM (Answer Relevance, Context Precision, Completeness, Coherency, Formatting Quality)

LLMs	Scores (Trial 1)	Scores (Trial 2)	Scores (Trial 3)
GPT-4o	(9.39, 8.98, 9.56, 9.33, 6.99)	(9.36, 8.94, 9.58, 9.32, 7.23)	(9.40, 8.99, 9.64, 9.38, 6.91)
GPT-4.1	(9.53, 9.03, 9.56, 9.34, 8.08)	(9.53, 9.01, 9.59, 9.33, 8.04)	(9.50, 8.98, 9.62, 9.34, 8.08)
DeepSeek-V3	(9.52, 9.01, 9.20, 9.20, 8.30)	(9.53, 9.03, 9.13, 9.18, 8.29)	(9.54, 9.03, 9.17, 9.23, 8.34)
DeepSeek- R1	(9.61, 9.18, 9.02, 9.26, 8.19)	(9.61, 9.18, 9.11, 9.27, 8.17)	(9.61, 9.19, 9.11, 9.28, 8.27)

A.9 Detailed Response Times for Three Questions Across Different LLMs in Three Trials

Question1: What is the appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA)?
Question2: What is the appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA)?
Question3: What is the appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA)?

Table A.3: Detailed Response Times for Three Questions Across Different LLMs in Three Trials

LLM	Trial	Time Required for Three Questions (Second)	Total Time Required for Three Questions (Second)
GPT-4o	Trial 1	(3.637, 3.447, 3.670)	10.754
	Trial 2	(3.670, 10.889, 5.898)	20.457
	Trial 3	(4.745, 3.114, 2.698)	10.557
GPT-4.1	Trial 1	(4.081, 3.661, 13.284)	21.026
	Trial 2	(5.260, 3.696, 4.513)	13.469
	Trial 3	(4.804, 3.441, 4.370)	12.615
DeepSeek-V3	Trial 1	(16.820, 18.376, 19.577)	54.773
	Trial 2	(23.568, 21.729, 22.502)	67.799
	Trial 3	(15.920, 21.088, 22.271)	59.279
DeepSeek-R1	Trial 1	(35.675, 34.207, 47.968)	117.85
	Trial 2	(40.092, 31.748, 33.087)	104.927
	Trial 3	(31.509, 33.885, 32.632)	98.026

A.10 Case Study: Incorrectly Answered Questions using Table Data (Chunk Size 512 and Overlap 50)

Table A.4: Incorrectly Answered Questions using Table Data (Chunk Size 512 and Overlap 50)

Incorrectly Answered Questions	Incorrect Responses	Reason for incorrectness
What is the appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cork powder?	The appropriate nozzle diameter for 3D printing using fused d eposition modeling with polylactic acid (PLA) mixed with cork powder is 0.4 mm, as per the document \"Enhanced_Table_Data.html\".	This value should be 0.8 mm.
What is the appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with Carbon Black?	I do not know. The document \"Enhanced_Table_Data.html\" from the 'Enhanced Table Data' folder does not provide information about the printing speed for PLA mixed with Carbon Black.	This value should be 30 mm/s.

A.11 Case Study: Incorrectly Answered Questions using Table Data (Chunk Size 512 and Overlap 100)

Table A.5: Incorrectly Answered Questions using Table Data (Chunk Size 512 and Overlap 100)

Incorrectly Answered Questions	Incorrect Responses	Reason for incorrectness
What is the appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC)?	The appropriate printing speed for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC) is 50 mm/s for one configuration and 60 mm/s for another conf iguration. \n\nSource: Enhanced_Table_Data.html	This should only be 60 mm/s.
What is the appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC)?	The appropriate layer heights for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC) are 0.2 mm and 0.38 mm. \n\nSource: Enhanced_Table_Data.html	This should only be 0.38 mm.
What is the appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC)?	The appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with microcrystalline cellulose (MCC) is 40°C for one configuration and 60°C for another configuration.\n\nSource: Enhanced_Table_Data.html	This should only be 60°C.
What is the appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC)?	The appropriate nozzle temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC) is 210°C or 250°C, as indicated in the document [Enhanced_Table_Data.html].	This should be 230°C.
What is the appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC)?	The appropriate layer height for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC) is 0.2 mm or 0.5 mm, as indicated in the document [Enhanced_Table_Data.html].	This should only be 0.2 mm.
What is the appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC)?	The appropriate nozzle diameter for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with cellulose nanocrystal (CNC) is 0.4 mm or 1.5 mm, as indicated in the document [Enhanced_Table_Data.html].	This should be 0.35 mm.
What is the appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with Hydroxyapatite (HA)?	The \"Enhanced Table Data\" document does not provide information on the appropriate bed or platform temperature for 3D printing using fused deposition modeling with polylactic acid (PLA) mixed with Hydroxyapatite (HA).	This should be 50°C.