LLM CUSTOMIZATION WITH WATSONX.AI: TWEAKING A MODEL FOR SPECIFICS TASKS

Tatiana Corina Dosescu, Assist. Prof., PhD, Titu Maiorescu University
Daniela Joita, Assoc. Prof., PhD**,** Titu Maiorescu University
Geanina Silviana Banu, Assoc. Prof., PhD**,** Titu Maiorescu University
Mironela Pirnau, Assoc. Prof., PhD, Titu Maiorescu University
Tudor Catalin Apostolescu, Assoc. Prof., PhD**,** Titu Maiorescu University
Claudia Elena Dinuca, Assist. Prof., PhD**,** Titu Maiorescu University
https://doi.org/10.66793/titui19proceeding1

Abstract: *This paper explores the customization of Large Language Models (LLMs) using IBM Watsonx.ai, focusing on prompt engineering, prompt tuning, supervised fine-tuning, and structured evaluation. Large Language Models (LLMs) have become foundational components in modern AI systems due to their ability to generalize across a wide range of language-driven tasks. However, the deployment of generic LLMs in real-world enterprise environments often requires substantial adaptation to satisfy domain-specific constraints, regulatory requirements, and performance expectations. Our study evaluates the role of Watsonx.ai's Granite foundation models—optimized for accuracy, transparency, and compliance with AI governance frameworks such as the EU AI Act. Overall, this paper highlights Watsonx.ai as a comprehensive platform for secure, compliant, and scalable LLM customization, enabling organizations to transform generic foundation models into highly specialized AI assets.*

1. INTRODUCTION

Large Language Models (LLMs) have revolutionized AI by enabling generative capabilities across text, code, and multimodal applications. However, base LLMs are generic and may lack sensitivity to domain-specific terminology and reasoning patterns.

IBM Watsonx.ai offers a comprehensive platform for modifying LLM behaviour through prompt engineering, prompt tuning, and supervised fine-tuning. LLMs enable natural communication with machines, supporting tasks like summarizing text, writing code, and drafting documents. They mark a major advance in AI, moving beyond keyword-based algorithms to deeper comprehension.

Widely used across industries, LLMs power tools such as OpenAI's ChatGPT, Anthropic's Claude, Microsoft's Copilot, Meta's Llama, Google's Gemini, and IBM's Granite models on watsonx.ai.

Training of the Large Language Models (LLMs) begins with massive text datasets (books, articles, websites, code, etc.) that are cleaned and preprocessed to remove errors and unwanted content. The text is tokenized into smaller units (tokens) such as words or subwords, allowing the model to learn language in a standardized way. LLMs are trained through self-supervised learning, which doesn't require labelled data. The model learns to recognize patterns and relationships by predicting missing information and optimizing its performance against an inferred "ground truth." Large language models use transformer architectures, introduced in 2017, which rely on the self-attention mechanism — the key innovation that allows the model to focus on different tokens depending on context. Self-attention helps the model understand relationships between distant tokens and supports parallel processing, enabling the training of extremely large datasets.

Each token is converted into a numerical vector called an embedding, which is gradually refined across multiple neural network layers to capture deeper contextual meaning. Positional encodings indicate the token's place in the sequence. For attention calculations, every embedding is transformed into three vectors — *query*, *key*, and *value*.

- The *query* represents what a token is looking for,
- The *key* represents the information each token holds, and
- The *value* provides that information, weighted by attention scores.

By comparing queries and keys, the model computes *attention weights* that determine how much each token influences others. This mechanism efficiently captures semantic associations (e.g., "dog" and "bark") and reduces the impact of irrelevant context. Through repeated *training*, the model optimizes billions or trillions of internal *parameters* using backpropagation, learning to predict language patterns such as grammar, facts, and reasoning.

After initial pretraining, *LLMs can be fine-tuned* to perform better in specific contexts. For example, a general-purpose foundational model can be fine-tuned on a dataset of legal questions and answers to create a *specialized legal chatbot*. Fine-tuning adapts a pretrained model to a particular domain, task, or style.

Experts may use one or a combination of several fine-tuning methods depending on the desired outcome. Fine-tuning usually occurs in a *supervised learning* setting, using a much smaller *labelled dataset*. The model updates its weights to align more closely with the new *ground truth*.

While *pretraining* gives the model broad general knowledge, *fine-tuning* adapts it to *specific tasks* such as: summarization, classification, and customer support.

This process allows the model to perform targeted functions with less data and computing power than training from scratch. Supervised fine-tuning is also useful for domain-specific customization, such as training on medical documents to enable accurate health-related responses. Standard supervised fine-tuning teaches models to mimic examples but doesn't promote advanced, multi-step reasoning. Because such tasks often lack labelled data, reinforcement learning is used to train reasoning models — LLMs capable of breaking down complex problems into smaller reasoning traces before generating an answer. These methods enable chain-of-thought reasoning and multi-step decision-making abilities.

2. Watsonx.ai Ecosystem for LLM Adaptation

Watsonx.ai integrates Prompt Lab, Tuning Studio, and Evaluation Studio in a unified interface. Granite models include variants for text generation, code generation, and multilingual reasoning. These models are optimized for enterprise governance, transparency, and compliance with regulations like the EU AI Act. Watsonx.ai integrates Prompt Lab, Tuning Studio, and Evaluation Studio in a unified interface. Granite models include variants for text generation, code generation, and multilingual reasoning. These models are optimized for enterprise governance, transparency, and compliance with regulations like the EU AI Act.

Instruction tuning improves an LLM's ability to follow human instructions. Training data consists of prompt-like tasks paired with ideal responses, helping the model better align with user intent. Since pretrained LLMs aren't naturally optimized for instruction following, this process enhances conversational and task-specific performance. After training, an LLM processes prompts through the following inference steps:

1. Tokenizes the input.
2. Converts tokens into embeddings.
3. Uses a transformer to generate output token by token.
4. Calculates probabilities for all possible tokens.
5. Selects the most likely token until completion.

The model doesn't "know" the final answer in advance; it predicts each token sequentially based on learned statistical relationships.

Prompt engineering is the quickest way to adapt a general-purpose LLM for domain-specific outputs, requiring no retraining.

Example:

"Respond in the style of a trained medical professional."

This helps generate more relevant results (though LLMs shouldn't be used for medical advice).

LLMs have other strategies to control their outputs:

- *Temperature:* controls the randomness of generated text.
- *Top-k / Top-p sampling:* limits token selection to the most probable ones, balancing creativity and coherence.

The *context window* is the maximum number of tokens the model can "see" at once. Early LLMs had short windows. Modern ones handle *hundreds of thousands of tokens*, enabling: full-document summarization, large-scale code assistance and long, continuous conversations.

RAG connects pretrained models to *external knowledge bases* for more accurate, up-to-date responses. Retrieved information is inserted into the context window, letting the model use it without retraining.

Example:

A weather-connected LLM can provide real-time forecasts by accessing a weather database instead of being retrained on meteorological data.

Building an LLM from scratch is resource-intensive, requiring vast datasets, GPUs, energy, and expertise — usually managed by large tech companies. However, many pretrained LLMs are available to developers via APIs for creating chatbots, information retrieval systems, and automation tools. Open-source models can also be deployed locally or in the cloud, with platforms like *GitHub, Hugging Face,* and *Kaggle* making AI development widely accessible. LLMs serve as the foundation for diverse AI applications. Emerging *AI agents* go beyond text generation by integrating memory, APIs, and logic to perform real-world tasks such as booking flights or controlling autonomous systems.

Common Use Cases of LLMs

- *Text Generation:* writing emails, articles, or legal briefs.
- *Summarization:* condensing long documents, reports, or customer histories.
- *AI Assistants:* providing real-time customer support through chatbots.
- *Code Generation:* assisting developers with code writing, debugging, and translation.
- *Sentiment Analysis:* interpreting customer feedback at scale.
- *Machine Translation:* delivering fluent multilingual communication.

- *Reasoning:* solving problems, planning steps, and explaining complex ideas clearly.

While powerful, LLMs face challenges such as:

- *Hallucinations:* generating false but convincing information.
- *Bias:* reproducing or amplifying prejudices from training data.
- *High resource consumption:* requiring significant compute power and energy.

These risks can be mitigated through strong *AI governance* — policies ensuring safety, ethics, and fairness.

Evaluation benchmarks provide quantitative scores for comparing models across multiple dimensions, including: a*ccuracy, efficiency, safety, fairness* and *robustness*

Red-teaming tests are used to reveal unsafe or biased behaviours.

Granite™ is IBM's flagship family of *foundation large language models (LLMs).* The *Granite 3.0* collection includes both *base pretrained* and *instruction-tuned* models with *2B* and *8B parameters*.

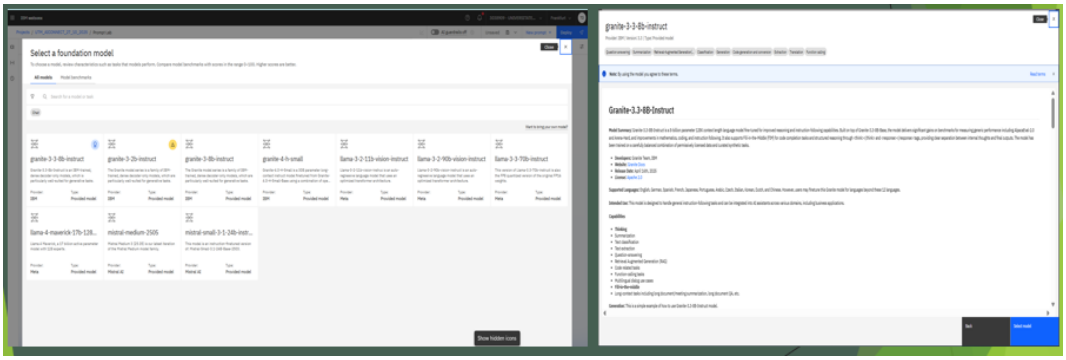

Fig. 1 Selection and exploration of an AI foundation model (specifically the *Granite 13B Instruct model*) within the *IBM watsonx.ai* platform

These open-source models perform strongly across: language understanding and generation, enterprise applications like cybersecurity, AI agent use cases involving tool or function calling, and retrieval-augmented generation (RAG) tasks, where models integrate external factual data for more accurate and current responses.

Granite 3.0 models are available for commercial use across multiple platforms, including *IBM watsonx™* portfolio products.

3. Prompt Engineering Techniques

Instruction tuning is a fine-tuning technique where large language models (LLMs) are trained on labelled datasets consisting of instruction-style prompts and their corresponding responses. This process improves a model's ability not only to perform specific tasks but also to follow human instructions more effectively, making pretrained models more practical for real-world use. Instruction tuning is a subcategory of fine-tuning used to adapt foundation models to downstream tasks, such as: *style customization, expanding core knowledge or vocabulary, and optimizing performance for specific applications*.

Instruction tuning often works alongside other fine-tuning techniques, such as *RLHF (Reinforcement Learning from Human Feedback)* for improving helpfulness and honesty, and *domain-specific fine-tuning* (e.g., programming data) to enhance specialized knowledge.

Pretrained LLMs are not inherently optimized for *following instructions or engaging in conversations* — they simply extend text sequences rather than truly responding to prompts. Instruction tuning helps make generated text more relevant and useful to users.

During *pretraining*, autoregressive models like Meta's Llama 2, OpenAI's GPT, Google's Gemini, or IBM's Granite are trained through *self-supervised learning* to predict the *next word* in massive text datasets. While this gives them strong linguistic coherence, it doesn't align their behaviour with *user intent*. For example, without fine-tuning, a base model might answer "Teach me how to make bread" with "in a home oven" — grammatically correct, but unhelpful.

Since retraining an LLM from scratch is impractical due to its billions of parameters, *fine-tuning* offers a more efficient solution. Techniques like *PEFT (Parameter-Efficient Fine-Tuning)* — including *partial fine-tuning* and *LoRA (Low-Rank Adaptation)* — allow adaptation with minimal computational cost.

Instruction tuning specifically uses supervised learning on labelled (input, output) pairs, where inputs resemble real user prompts and outputs represent desired responses. This helps the model learn to follow instructions directly. Instruction tuning bridges the gap between:

- the model's original goal — *predicting the next word*, and

- the user's goal — *getting a useful, instruction-following response*.

It makes model behaviour *more practical, predictable, and aligned* with real-world applications.

Instruction tuning fine-tunes LLMs on labelled datasets containing diverse instruction-following tasks, improving their general ability to follow human instructions. This reduces the need for complex prompt engineering and few-shot examples. Instruction datasets can be created by humans or generated by other LLMs.

Each training example typically includes:

1. *Instruction* – a natural language command (e.g., "Translate this sentence from English to Spanish.")
2. *Additional context* – optional supporting text relevant to the task.
3. *Desired output* – the correct answer serving as the *ground truth* for model optimization.

Adding more instruction-based tasks to the dataset improves performance even on unseen tasks, demonstrating that instruction tuning enhances the model's overall ability to follow instructions.

Parameters in large language models are settings that control and optimize the model's output and behaviour.

Modern LLMs can have *billions of parameters*. Smaller models have fewer parameters—making them lighter and faster but less capable of capturing complex relationships.

- *Trainable parameters* (weights, biases) determine how the model processes data.
- *Hyperparameters* are external settings that guide learning and shape the model's output. Open-source models often expose these settings, enabling customization for specific tasks.

Key hyperparameters include:

- *Number of layers:* Controls model depth and complexity. Too many → overfitting; too few → limited learning capacity.
- *Context window:* Defines how many tokens the model can "see" at once. Larger windows improve coherence and accuracy but raise computational cost.
- *Temperature:* Controls randomness and creativity in text generation. Low values → factual and consistent; high values → creative but less predictable.
- *Top-p (nucleus sampling):* Limits token selection to the most probable ones within a cumulative probability threshold *p*.
- *Top-k:* Restricts generation to the *k* most likely tokens.
- *Max tokens:* Sets the maximum output length.
- *Learning rate:* Determines how quickly model weights adjust during training.
- *Frequency, repetition, and presence penalties:* Reduce redundancy and encourage lexical diversity.
- *Stop sequences:* Token patterns that halt text generation (e.g., a period or keyword).

Several techniques improve internal parameter efficiency and performance:

- *Fine-tuning:* Adjusts weights and biases for specific tasks.
- *PEFT (Parameter-Efficient Fine-Tuning):* Updates only a small subset of parameters.
- *Transfer learning:* Reuses pretrained knowledge for new tasks.
- *Quantization:* Simplifies internal math for smaller, faster models with minimal accuracy loss.
- *Early stopping:* Prevents overfitting by halting training when improvements plateau.

When working with LLMs like *IBM Granite Instruct* several parameters can be adjusted to control randomness and diversity in generated text.

The three most common parameters are:

- *do_sample* – Enables random sampling during generation. When set to *True*, the model selects tokens probabilistically rather than always choosing the most likely next word. This must be enabled to adjust *temperature*.
- *top_k* – Limits random sampling to the *k most probable tokens*, preventing the model from picking highly unlikely words while still allowing variability.
- *top_p (nucleus sampling)* – Limits token selection to a *cumulative probability threshold* (e.g., 0.95), so the model samples only from tokens that together make up 95% of total probability. This balances creativity and coherence.

Together, these parameters control how diverse, coherent, or predictable the model's responses are, helping fine-tune the balance between creativity and accuracy in text generation.

Many LLMs provide parameters that allow users to manage and shape the model's output more precisely.

Key output control parameters include:

- *Maximum length*: Sets the total number of tokens the model can generate, helping to keep responses concise and relevant.
- *Stop sequences:* Define specific token patterns that signal the model to stop generating text (e.g., "Best regards," in an email). Useful for structured outputs such as emails, lists, or dialogues.
- *Frequency penalty:* Reduces the likelihood of repeating the same words by penalizing tokens based on how often they appear, resulting in more varied and natural text.

- *Presence penalty:* Similar to the frequency penalty, but applies a penalty simply for the reappearance of a token, regardless of how often it has been used. This encourages greater lexical diversity.

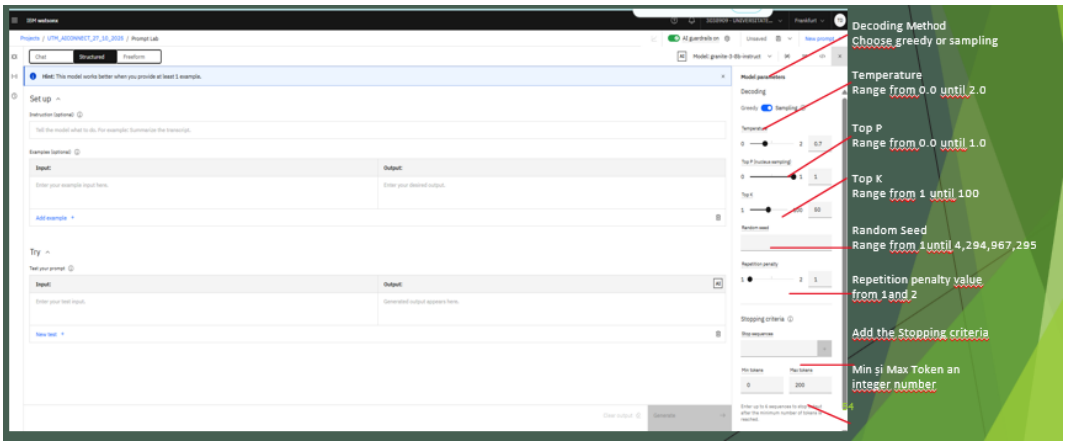

Fig. 2 This section allows users to customize the model's behavior (for example, the length, coherence, or creativity of its responses) when testing prompts in the *Prompt Lab* environment within ***IBM watsonx.ai***.

Example — Influence of Temperature on Text Generation (IBM Granite 3.1, 2B parameters)

To illustrate the impact of the temperature parameter on generative behaviour, experiments were conducted using the IBM Granite 3.1 model with 2 billion parameters. The same input prompt was used while varying the temperature values to observe differences in coherence, creativity, and narrative structure. To observe how temperature influences text generation, IBM Granite 3.1 (2B parameters) was given the prompt:

"Write a story about a data scientist who loves Python."

- Temperature = 0.1:

Once upon a time, in the bustling city of San Francisco, there lived a brilliant data scientist named Alex. Alex was known for their exceptional skills in Python programming and data analysis. They spent their days and nights immersed in the world of data, using Python to extract insights and make predictions.

One day, Alex was tasked with analysing customer churn for a popular streaming service. They used Python's scikit-learn library to build a machine learning model that could predict which customers were likely to cancel their subscriptions.

The story is factual, structured, and predictable — focused on data analysis and machine learning tasks using Python.

- Temperature = 0.75:

Once upon a time, in the bustling city of San Francisco, there lived a brilliant data scientist named Alex. Alex was known for their deep love for Python, a programming language that they believed was the perfect tool for data analysis and machine learning.

Every day, Alex would wake up early, brew a pot of coffee, and sit down at their desk. They would spend hours writing code in Python, exploring datasets, and building predictive models. Their love for Python was not just about the language itself, but also about the community that had grown around it.

The story becomes more expressive and natural, adding personal details about Alex's passion for Python and daily routine.

- Temperature = 1.25:

Once upon a time, in a bustling city filled with towering skyscrapers and neon lights, there was a data scientist who loved Python more than anything in the world. The data scientist has a feline helper named Suki. Suki is a curious and playful cat, always eager to explore every nook and cranny of the house.

One day, the data scientist is working on a complex coding problem. Suki keeps meowing loudly as if she is trying to communicate a message to the data scientist. The data scientist notices that the loud meowing usually happens when they are close to a solution to the complex coding problem.

The story grows imaginative and whimsical — introducing a cat companion and creative narrative elements.

We observe that as temperature increases, the generated text becomes: more varied and creative, richer in details and storytelling, but also less predictable and more prone to randomness.

4. LLM Customization

LLM customization is the process of adapting a *pretrained foundation model* to meet specific use-case requirements. It tailors a general-purpose LLM into a *task-specific model*.

- *Data Preparation*

High-quality, domain-specific data is collected to improve the model's accuracy and relevance.

- *Model Selection*

Choosing an appropriate model depends on factors like size, architecture, resource demands, and project goals.

- *Model Customization*

ML specialists adjust the foundation model to better handle downstream tasks, tuning its behaviour to fit specific needs.

- *Iteration*

The model is fine-tuned and improved step by step based on performance feedback from each training cycle.

- *Testing*

Before deployment, the model is thoroughly tested to ensure reliability and to avoid issues like catastrophic forgetting (losing previously learned knowledge).

- *Deployment*

The final customized model is deployed into a production environment (e.g., an AI-based application or API), ready for real-world use. Depending on the use case and desired output, developers and ML specialists can choose among several methods to customize large language models (LLMs) for specific downstream tasks.

The *main customization techniques* are: *Retrieval-Augmented Generation (RAG), Fine-tuning and Prompt Engineering*.

- *RAG* connects an LLM to an *external data source* to extend its knowledge base. When a user submits a query, *RAG* retrieves relevant information from a connected database and combines it with the user's input, giving the model additional context before generating a response. *RAG Works:*

1. *Prompting* – The user enters a query.
2. *Querying* – The system converts the query into an *embedding* and searches for similar data in a *vector database*.
3. *Retrieval* – Relevant information is extracted.
4. *Generation* – Retrieved data is combined with the user's query and sent to the LLM for response generation.
5. *Delivery* – The final answer is returned to the user.

The name "Retrieval-Augmented Generation" comes from the process of retrieving external data to augment the model's output. By integrating external knowledge sources, RAG enables models to: provide more accurate and reliable answers, reduce the need for costly retraining and leverage existing structured or unstructured data.

This approach is especially effective for domain-specific applications and can enhance small language models (SLMs) by combining their efficiency (low cost, fast inference, minimal compute requirements) with RAG's contextual accuracy.

Thus, a RAG-based Q&A system can access a curated knowledge base to deliver highly precise answers without retraining the underlying model.

- *Fine-tuning* involves iterative adjustments to a model's internal *parameters (weights and biases)* to improve performance on specific tasks. Modern methods focus on adjusting *only a subset* of parameters or adding lightweight components. Notable fine-tuning methods include:
  - *Parameter-Efficient Fine-Tuning (PEFT)***:** freezes most parameters and updates only a small portion, saving resources.
  - *Transfer Learning***:** applies knowledge from pretraining to related new tasks (e.g., adapting an image classifier to new object types).
  - *Low-Rank Adaptation (LoRA):* adds small, trainable low-rank matrices to modify model behaviour for new use cases without overwriting core parameters.
  - *Reinforcement Learning with Human Feedback (RLHF):* uses human ratings to train a reward model that aligns the LLM's responses with human preferences.
  - *Continual Fine-Tuning (CFT):* incrementally adapts the model to new data and tasks while avoiding *catastrophic forgetting*.

The benefits: Fine-tuning customizes models for specific applications, saving time and cost versus training from scratch. Modern approaches require less data and computing power.

- *Prompt Engineering* also called *in-context learning* or *prompt-based learning*, this technique improves output by crafting *clear, context-rich prompts*.

During inference, users can provide structured instructions or examples to guide the model's behaviour.

Example:
A summarization prompt that includes a sample "bulleted list" can help the model mimic that format in its response.

Common techniques:

- *Few-shot prompting:* includes a few examples of correct outputs.
- *Chain-of-thought (CoT) prompting***:** guides the model to reason step-by-step toward a solution.

Advantages:
Prompt engineering requires *no code or retraining*, only creativity and understanding of the model's logic. It empowers users—even non-programmers—to effectively adapt open-source LLMs for specialized applications.

In practice, it serves as an *accessible entry point to AI and NLP*, rewarding experimentation and thoughtful design.

CONCLUSIONS

This paper demonstrated that *IBM watsonx.ai* provides a coherent, auditable, and scalable framework for *customizing Large Language Models (LLMs)* to perform domain-specific tasks. The platform integrates three complementary adaptation layers — *Prompt Engineering*, *Prompt Tuning*, and *Supervised Fine-Tuning (SFT)* — enabling flexible control, efficient parameter optimization, and deep domain alignment. At the core of these workflows lies the *Granite family of foundation models*, optimized for enterprise-grade requirements such as transparency, safety, reproducibility, and regulatory compliance (e.g., *EU AI Act, GDPR*). Together with *Prompt Lab, Tuning Studio*, and *Evaluation Studio, watsonx.ai* enables an iterative improvement cycle — design → training → evaluation → deployment — with full traceability and performance monitoring.

The main outcome of this study is that the choice of customization strategy should be *guided by constraints and objectives:*

- *Prompt Engineering* is ideal when speed, flexibility, and zero retraining are priorities — suitable for prototyping, formatting control, and stylistic consistency.
- *Prompt Tuning (PEFT)* is recommended when domain alignment and low computational cost are key. It enables rapid adaptation with minimal resource consumption and negligible performance degradation.
- *Supervised Fine-Tuning (SFT)* is reserved for cases demanding substantial accuracy gains and domain-specific reasoning, where higher costs and larger labelled datasets are justified.

The study also emphasized the role of Retrieval-Augmented Generation (RAG) as a complementary technique that extends a model's knowledge without retraining, reducing hallucinations and improving factuality. At the same time, *AI governance* mechanisms — including audit trails, bias evaluation, drift detection, and access control — must accompany the entire model lifecycle to ensure safe, ethical, and transparent deployment.

Limitations and risks include dependency on *data quality*, trade-offs between *creativity and predictability* (e.g., via temperature or sampling parameters), *annotation costs* for SFT, and the necessity of a *strong ethical framework* for high-stakes applications such as finance, healthcare, or law.

Practical recommendations suggest starting with *Prompt Engineering* and basic safety guardrails, proceeding to *Prompt Tuning* for internal alignment, and adopting *SFT* only when advanced reasoning or accuracy demands justify the investment. Using *Evaluation Studio* for multi-criteria benchmarking (accuracy, robustness, safety) and enforcing *AI governance* ensures responsible use. Within this ecosystem, *watsonx.ai* and *the Granite models* enable organizations to transform general-purpose LLMs into *specialized, production-ready AI assets.*

5. REFERENCES

1. Zhao, Fengxiang, et al. "A new method using LLMs for keypoints generation in qualitative data analysis." 2023 IEEE Conference on Artificial Intelligence (CAI). IEEE, 2023.

2. Carneros-Prado, David, et al. "Comparative study of large language models as emotion and sentiment analysis systems: A case-specific analysis of GPT vs. IBM Watson." International Conference on Ubiquitous Computing and Ambient Intelligence. Cham: Springer Nature Switzerland, 2023.

3. Li, H., Wang, S. X., Shang, F., Niu, K., & Song, R.. Applications of large language models in cloud computing: An empirical study using real-world data. Spectrum of Research, 4(1) (2024).

4. https://www.ibm.com/think/topics/llm-customization#692473880

5. https://www.ibm.com/think/topics/large-language-models#692473873