AutoChemSchematic AI: A Closed-Loop, Physics-Aware Agentic Framework for Auto-Generating Chemical Process and Instrumentation Diagrams

Sakhinana Sagar Srinivas*
sagar.sakhinana@tcs.com
Tata Research
Bangalore, India

Shivam Gupta*
shivam.gupta@tcs.com
Tata Research
New Delhi, India

Venkataramana Runkana
venkat.runkana@tcs.com
Tata Research
Pune, India

Abstract

Recent advancements in generative AI have accelerated the discovery of novel chemicals and materials; however, transitioning these discoveries to industrial-scale production remains a critical bottleneck, as it requires the development of entirely new chemical manufacturing processes. Current AI methods cannot auto-generate Process Flow Diagrams (PFDs) or Piping and Instrumentation Diagrams (PIDs)—despite their critical role in scaling chemical processes—while adhering to engineering constraints. We present a closed-loop, physics-aware framework for the automated generation of industrially viable PFDs and PIDs. The framework integrates domain-specialized small-scale language models (SLMs) (trained for chemical process QA tasks) with first-principles simulation, leveraging three key components: (1) a hierarchical knowledge graph of process flow and instrumentation descriptions for 1,020+ chemicals, (2) a multi-stage training pipeline that fine-tunes domain-specialized SLMs on synthetic datasets via Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), and Retrieval-Augmented Instruction Tuning (RAIT), and (3) DWSIM-based simulator-in-the-loop validation to ensure feasibility. To improve both runtime efficiency and model compactness, the framework incorporates advanced inference-time optimizations—including FlashAttention, Lookahead Decoding, PagedAttention with KV-cache quantization, and Test-Time Inference Scaling—and independently applies structural pruning techniques (width and depth) guided by importance heuristics to reduce model size with minimal accuracy loss. Experiments demonstrate that the framework generates simulator-validated process descriptions with high fidelity, outperforms baseline methods in correctness, and generalizes to unseen chemicals. By bridging AI-driven design with industrial-scale feasibility, this work significantly reduces R&D timelines from lab discovery to plant deployment.

1 Introduction

Process Flow Diagrams (PFDs) and Piping and Instrumentation Diagrams (PIDs) are standard engineering diagrams used in the chemical process industry. A PFD provides a high-level schematic of the flow of materials and energy through a chemical production process. It depicts major equipment, process streams, and key operating conditions for specific units within the process, without detailing the instrumentation or control systems used during plant operation. PIDs, which build upon PFDs, offer a more detailed schematic of the instrumentation and control systems essential for monitoring, operational control, safety, and maintenance of the chemical process plant. The purpose of a PFD (see Figure 1) is to show what happens in the process (i.e., the key physical or chemical transformations) and where it happens (i.e., in which major equipment units), rather than how the process is controlled or operated. The purpose of a PID (see Figure 2) is to illustrate how the process operates and is controlled—via valves, sensors, and control loops—not just what happens or where it occurs. PFDs and PIDs serve as foundational documents for chemical process simulations, which in turn drive the development of digital twins. These digital twins integrate first-principles or data-driven models with real-time sensor and actuator data, enabling dynamic monitoring, predictive control, and AI-driven automation for closed-loop process optimization. Recent advancements in generative AI are transforming chemical and materials science [7, 16, 20, 23, 34, 47, 55, 58, 58, 61] by accelerating the autonomous discovery of environmentally sustainable, next-generation specialty chemicals and the development of low-cost, high-performance, materials-based products. Simultaneously, these advancements are streamlining complex first-principles simulation workflows and reducing reliance on tedious lab-based trial-and-error, enabling faster innovation, lower R&D costs, and more sustainable product innovation. However, realizing the full potential of these discoveries requires the development of new production processes to transition them from computer simulations or wet-lab experiments to industrial-scale production—a significant challenge in bringing better products to market more rapidly. Current methods [2, 14, 19, 42, 48, 54] are not designed to auto-generate process flow schematics (e.g., PFDs) or instrumentation and control layouts (e.g., PIDs) for novel industrial-scale chemical production processes, significantly limiting their utility. Furthermore, these approaches overlook essential process context: for PFDs, this includes the high-level objectives—what the process achieves and in what sequence—while for PIDs, it requires details on how the process is operated, monitored, and controlled. As a result, they cannot justify design choices or the control and instrumentation logic needed for safe, stable, and efficient plant operations. Additionally, existing methods fail to integrate first-principles-based simulators to verify the physical and operational feasibility of generated PFDs and

*Both authors contributed equally to this research.

PIDs, further compromising industrial reliability. Moreover, the manual, expertise-intensive creation of novel PFDs and PIDs creates a bottleneck that hampers simulation fidelity, digital twin accuracy, and scalable AI deployment. To address these limitations, we present a closed-loop, self-driving lab framework for the auto-generation of high-fidelity process flow and instrumentation descriptions, accelerating the development of novel chemical processes. Implemented as an enterprise-grade, cloud-based SaaS solution, our framework significantly expedites the simulation-to-lab-to-pilot-to-plant scale-up pipeline, ensuring that only industrially viable, sustainable, and efficient processes advance to commercialization. The platform serves as an end-to-end process schematics modeling tool, automating design, simulation, and optimization with minimal human intervention. By integrating first-principles, physics-aware modeling with iterative reflection and adaptive learning, the framework continuously self-improves, enhancing the reliability of AI-generated process schematics and control strategies. To address these challenges, we propose a framework integrating three key innovations: (1) hierarchical knowledge graphs for retrieval-augmented generation, (2) domain-specialized small-scale language models (SLMs) fine-tuned through multi-stage training, and (3) physics-aware simulator validation. By combining these components in a closed-loop system, our approach maintains generative flexibility while ensuring industrial feasibility. In the following section, we formalize this methodology, beginning with data curation and knowledge graph construction, then proceeding to SLM training and simulation-based verification.

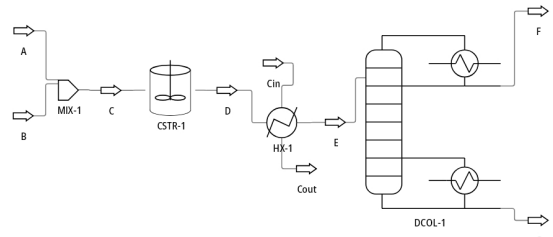


Figure 1: The figure shows a high-level schematic of a chemical process depicting material flow from reactant inlets (A and B) through a mixer (MIX-1), a continuous stirred-tank reactor (CSTR-1), a heat exchanger (HX-1), and a distillation column (DCOL-1), yielding product streams F and G. Major equipment and stream connections are illustrated, excluding instrumentation and control logic. This abstraction facilitates understanding of core process operations and transformations.

2 Methodology

Our methodology integrates data curation, knowledge graph construction, advanced language model fine-tuning, retrieval augmentation, inference optimization, and engineering validation to create specialized and efficient SLMs for chemical process engineering tasks—specifically the interpretation, analysis, and generation of PFDs and PIDs. The pipeline begins with the curation of the ChemAtlas database, comprising over

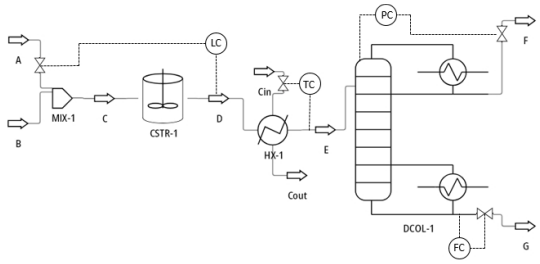


Figure 2: The figure shows the detailed PID of a chemical process showing instrumentation and control systems, including: level control (LC) on reactor CSTR-1 regulating feed A; temperature control (TC) on column feed E adjusting HX-1 utility flow; pressure control (PC) at DCOL-1 overhead controlling product F; and flow control (FC) on bottoms product G. The diagram specifies control strategies and safety-critical parameters.

1,020 industrial chemicals extracted from manufacturer catalogs. An agentic web navigation framework autonomously retrieves, interprets, and synthesizes multimodal data from diverse sources to generate structured process flow and instrumentation descriptions for each chemical. This structured information is used to construct a knowledge graph (KG), where text chunks are processed by GPT-4o to extract semantic triples (subject–predicate–object). Entities are canonicalized based on high semantic similarity (via embeddings) and string similarity (via normalized Levenshtein distance), and the resulting graph is partitioned into hierarchical communities using the Leiden algorithm to optimize modularity for efficient retrieval. Leveraging ChemAtlas, we employ advanced teacher LLMs (GPT-4o and Anthropic Claude Haiku) to generate and cross-validate over 20,000 synthetic question–answer (QA) pairs through a self-instruct bootstrapping method, initialized from a small seed set of human-authored demonstrations. These QA pairs are rigorously scored, validated, and filtered using NVIDIA's Nemotron-4-340B reward model to ensure quality before training SLMs. The final dataset comprises six specialized synthetic subsets: (1) Factual QA, designed to reinforce foundational knowledge in chemical process engineering; (2) SynDIP, comprises comprehensive QA pairs covering process flow and instrumentation descriptions(equivalent to agentic web retrieved knowledge on industrial chemical production but generated from pretrained knowledge of base LLMs); (3) LogiCore, featuring multi-step reasoning chains for process design justification, control logic validation, and related tasks; (4) DPO, containing preference-labeled response pairs for alignment tuning via Direct Preference Optimization; (5) Local RAIT, supporting document-grounded QA generation using individual SynDIP process descriptions; and (6) Global RAIT, supporting QA that requires cross-document synthesis by integrating multiple SynDIP process descriptions. Additionally, we construct a 1.5K QA out-of-distribution benchmark dataset from ChemAtlas to evaluate the framework's generalization across core QA tasks. We also introduce the ChemEval dataset, consisting of 100 held-out chemicals, to assess the framework's zero-shot

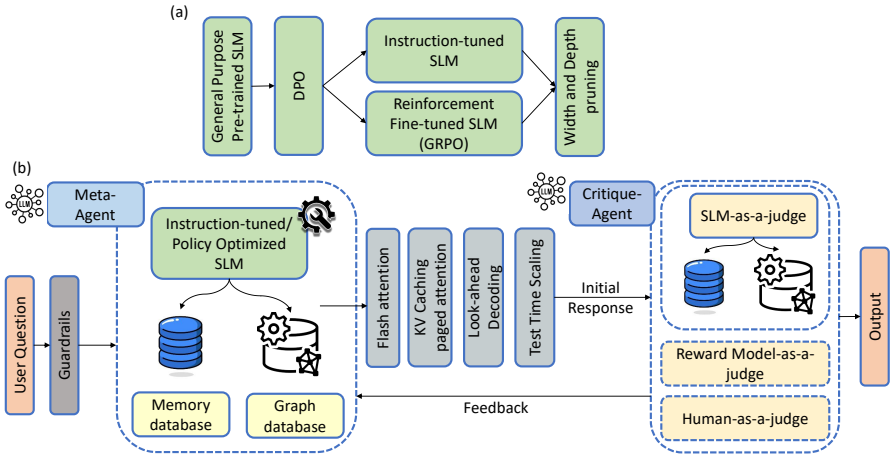


Figure 3: Overview of the integrated framework. (a) The SLM fine-tuning pipeline depicts initial DPO alignment followed by instruction tuning or GRPO reinforcement learning, concluding with optional width/depth pruning. (b) The operational RAG framework illustrates a Meta-Agent coordinating the specialized SLM (from part a), which accesses memory and graph databases for context. The SLM's inference is accelerated via optimizations (FlashAttention, Paged KV Caching, Lookahead Decoding, Test-Time Scaling). Generated responses are refined iteratively through a feedback loop managed by a Critique-Agent employing diverse judges (e.g., Nemotron-4-340B reward model, LLM-as-a-judge like GPT-4o/Haiku, or human evaluation).

generation performance for complete process flow and instrumentation descriptions of previously unseen chemicals. Base SLMs, specifically Llama-3.2-1B and SmolLM-135M, are customized using Quantized Low-Rank Adaptation (QLoRA) with frozen base weights. We employ two distinct fine-tuning strategies on the synthetic datasets. The first follows a sequential pipeline: Supervised Fine-Tuning (SFT) on the combined Factual QA, SynDIP, and LogiCore datasets; Direct Preference Optimization (DPO) on curated preference-labeled DPO datasets; and Retrieval-Augmented Instruction Tuning (RAIT) on Local and Global RAIT datasets. The second strategy adopts a reinforcement learning approach using Group Relative Policy Optimization (GRPO), applied first to the SFT datasets and then refined on RAIT datasets. This approach optimizes a composite reward function combining ROUGE-L F1 scores, length ratio penalties, and LLM-as-a-judge quality scores, stabilized by KL divergence regularization. The fine-tuned SLMs are integrated with the structured knowledge graph through a Graph RAG framework. During inference, the framework retrieves relevant graph communities by comparing query embeddings to pre-computed community summaries, dynamically retrieves top communities, and then constructs a subgraph containing interconnected entities, semantic relationships, and source text chunks, and leverages this context for grounded, multi-hop reasoning. To enhance performance, a suite of inference optimization and reliability techniques are implemented: structural pruning (width and depth) guided by importance heuristics reduces model size; PagedAttention combined with KV cache quantization mitigates memory fragmentation and reduces cache footprint; Lookahead Decoding accelerates generation latency through parallel token speculation; FlashAttention optimizes the core attention computation to reduce memory bandwidth bottlenecks; and Test-Time Inference Scaling improves output reliability using self-consistency sampling, confidence-weighted entropy scoring, iterative self-reflection/revision, and consensus aggregation. Finally, the practical engineering feasibility of generated process flow and instrumentation descriptions are validated using the DWSIM open-source chemical process simulator, where PFDs are translated into flowsheets to verify material/energy balances and thermodynamic consistency, while PIDs are functionally assessed by implementing control loops in DWSIM's dynamic environment to evaluate stability and control performance (e.g., setpoint tracking, disturbance rejection). This comprehensive methodology ensures the developed SLMs are knowledgeable, aligned with engineering principles, efficient, and reliable for practical deployment. Figure 3 visually outlines the overall architecture. Part (a) depicts the SLM fine-tuning pipeline, showing the progression from a general pre-trained model through initial preference alignment (DPO), followed by task-specific fine-tuning via either instruction tuning or reinforcement learning (GRPO), and concluding with model compression (pruning). Part (b) illustrates the operational RAG framework. It shows how a user query, after passing guardrails, is processed by a Meta-Agent. This agent utilizes the specialized SLM developed in part (a) as its core reasoning engine. The SLM, guided by the Meta-Agent, retrieves necessary context by accessing both a Memory database (e.g., for conversational history) and a Graph database (containing structured process knowledge). This retrieved information informs the SLM's

response generation. The inference process of the SLM is further enhanced by integrated optimizations (FlashAttention, Paged Attention KV Caching, Lookahead Decoding, Test-Time Scaling). An initial response generated by the SLM undergoes evaluation by a Critique-Agent, utilizing feedback mechanisms (SLM-as-a-judge, Reward Model-as-a-judge, or Human-as-a-judge) to potentially trigger refinement before producing the final Output. In summary, our integrated framework leverages knowledge graph-based retrieval augmentation, domain-specific SLM fine-tuning pipelines, comprehensive inference optimizations, and feedback-driven refinement. This approach yields robust performance on complex reasoning tasks and demonstrates effective generalization via the generation of plausible, simulator-validated process descriptions for previously unseen chemicals.

3 Experiments

3.1 Datasets

We curated a chemical database comprising over 1,120 compounds drawn from sectors such as electronics, energy storage, pharmaceuticals, advanced manufacturing, and utilities, with a focus on chemicals essential to modern industrial and technological applications. The data were programmatically extracted from product catalogs of leading manufacturers—including BASF, Dow Chemical, DuPont, Solvay, Mitsubishi Chemical, Bayer, Evonik, SABIC, and LyondellBasell—ensuring both reliability and broad industrial coverage. The dataset consists of two components. ChemAtlas: a core collection of 1,020 chemicals. For each chemical in ChemAtlas, we employ an AI-driven agentic web navigation framework that autonomously retrieves, interprets, and synthesizes data from diverse public sources to extract detailed descriptions of production processes. This structured data serves as the foundation for populating chemical knowledge graphs that support reasoning and retrieval in our Graph RAG framework, enabling LMs) to effectively interpret contextual information and answer user queries. To ensure consistency and correctness, we further use advanced large language models (LLMs)—specifically GPT-4o and Anthropic Claude Haiku—to generate and cross-validate chemical-specific production process descriptions derived from agentic web navigation, leveraging their pre-trained knowledge for automated validation. The second component, ChemEval, comprises a held-out evaluation set of 100 chemicals curated to rigorously test the framework's generalization capabilities in generating process flow and instrumentation descriptions for chemicals not present in ChemAtlas. Additionally, we adopt a teacher–student transfer learning approach by generating custom synthetic datasets from the ChemAtlas database. This includes 20,000 question–answer (QA) pairs created by teacher LLMs—specifically GPT-4o and Anthropic Claude Haiku—to train small language models (SLMs) such as LLaMA-3.2-1B and SmolLM-135M for domain-specific tasks involving the interpretation, analysis, and generation of process flow and instrumentation descriptions. A small seed set of human-authored instruction–response examples was used to initiate high-quality QA generation through iterative synthesis, guided by predefined templates and a self-instruct bootstrapping strategy. The generated outputs are scored, validated, and filtered using NVIDIA's Nemotron-4-340B reward model. The resulting datasets span a diverse range of QA pairs, covering factual knowledge, preference alignment, process flow and instrumentation interpretation, logical and multi-step chain-of-thought reasoning, sensor layout planning, comparative process analysis, and error detection and correction. These datasets are specifically designed to enhance the capabilities of SLMs in complex process engineering tasks, such as equipment and piping layout generation, sensor and instrumentation placement, and the understanding of interrelated system components. The curated dataset consists of six specialized synthetic subsets, each systematically constructed to induce and reinforce a specific functional capability in SLMs. The Factual QA dataset, constructed through hierarchical topic decomposition of chemical process engineering concepts, enhances factual recall and foundational domain knowledge. The SynDIP dataset contains QA pairs of process flow and instrumentation descriptions. The LogiCore dataset consists of multi-step reasoning QA pairs grounded in process flow and instrumentation descriptions. These pairs are crafted to justify process design choices, validate control logic, and explain flow sequencing within chemical process diagrams. The DPO dataset contains preference-labeled QA pairs, each comprising a preferred and a dispreferred response, distinguished using score differentials from a reward model. This structure supports alignment tuning via Direct Preference Optimization. The RAIT (Retrieval-Augmented Instruction Tuning) datasets aim to improve the SLMs' ability to effectively incorporate retrieved context into generation. Local RAIT, which grounds QA pairs in individual SynDIP-derived text chunks, enables precise, context-specific information extraction and answer generation. In contrast, Global RAIT utilizes semantically clustered groups of chunks—potentially spanning multiple SynDIP-derived documents—to foster the synthesis of information across related segments, thereby supporting cross-contextual reasoning and comprehensive understanding. In addition, we construct a 1.5K QA-pair out-of-distribution (OOD) benchmark dataset from ChemAtlas using a self-instruct approach with teacher LLMs (OpenAI o3 and o4-mini) to generate synthetic QA pairs. These pairs are iteratively created from agentic web-retrieved information and filtered for quality using reward mode to evaluate whether fine-tuned SLMs can generalize across core capabilities, including factual knowledge, reasoning, instruction following, and the interpretation and analysis of process flow and instrumentation tasks. For each chemical in ChemEval, the teacher models (GPT-4o and Claude Haiku) generated process flow and instrumentation descriptions in the form of QA pairs using the same self-instruct bootstrapping method. These QA pairs served as reference targets for quantitatively evaluating the framework's ability to generate accurate descriptions for unseen chemicals.

3.2 Experimental Setup

Graph Retrieval-Augmented Generation (Graph RAG) integrates structured knowledge graphs with large language models to enhance retrieval and reasoning. Our implementation begins with domain-specific documents—focused on chemical production processes—retrieved via autonomous agentic web navigation from the ChemAtlas database. The raw text is segmented into overlapping chunks using a sliding window approach, preserving local context while ensuring cross-chunk continuity. Each text chunk is processed by GPT-4o to extract subject-predicate-object triples, forming semantic edges between entity nodes in the knowledge graph. Structural containment edges link each entity back to its source chunk, maintaining provenance and traceability. To resolve redundancy, we apply a canonicalization step: entities are merged only if they exhibit both high semantic similarity (measured via text-embedding-3-small embeddings) and high string similarity (evaluated using normalized Levenshtein distance), with both metrics exceeding predefined thresholds. For efficient retrieval, we partition the graph into hierarchical communities using the Leiden algorithm [53], optimizing for modularity to ensure semantically coherent clustering. Prior to inference, each community is summarized by GPT-4o, and these summaries are embedded for fast similarity comparison. Given a query, the framework retrieves the top-K most relevant communities, dynamically constructing a subgraph that includes their interconnected entities, semantic relationships, and originating chunks. This structured context is then passed to the reasoning model, ensuring grounded, multi-hop generation. We fine-tuned the Llama-3.2-1B and SmolLM-135M models using Quantized Low-Rank Adaptation (QLoRA)[10], which adapts low-rank matrices to key transformer projection layers while maintaining the base model weights frozen in 4-bit NormalFloat (NF4) precision. All experiments employed identical training parameters: an 8-bit AdamW optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$), a learning rate of 2×10^{-4} with linear decay, weight decay of 0.01, and an effective batch size of 8 (achieved through a per-device batch size of 2 with 4 gradient accumulation steps), and a maximum sequence length of 4096 tokens enabled by gradient checkpointing. Training was conducted in BF16/FP16 mixed precision on NVIDIA H100 GPUs. We explored two fine-tuning strategies. The first strategy employed a sequential, multi-phase pipeline comprising three stages: (1) Supervised Fine-Tuning (SFT) on the combined training splits of the Factual QA, SynDIP, and LogiCore datasets for 15 epochs to integrate instruction-following capabilities and foundational domain knowledge into the SLMs; (2) Direct Preference Optimization (DPO) on the curated DPO dataset's training split for 5 epochs to align model outputs with human preferences; and (3) Retrieval-Augmented Instruction Tuning (RAIT) leveraging the Local and Global RAIT dataset's training splits for 15 epochs to improve the SLM's ability to produce contextually grounded responses. The second strategy employed the Group Relative Policy Optimization (GRPO) reinforcement learning algorithm [44], adapted for direct policy optimization. This approach progressed through two sequential stages: first fine-tuning the pretrained base model on the train splits of the combined Factual QA, SynDIP, and LogiCore datasets, then refining the resulting SFT checkpoint using the train splits of the Local RAIT and Global RAIT datasets. Both stages maintained the same QLoRA configuration described earlier. The optimization process maximized a clipped surrogate objective [41] conceptually similar to Proximal Policy Optimization (PPO), using normalized advantages derived from a composite reward function with three components: ROUGE-L F1 score (weight=0.3), a length ratio penalty encouraging similarity to reference response lengths (weight=0.2), and an LLM-as-a-Judge quality score assessing answer correctness and relevance (weight=0.5). Rewards and advantages were computed relative to groups of G=4 responses sampled from the policy for each input. Training stability was ensured through β-weighted KL divergence regularization against the relevant reference policy (either the pretrained base model or SFT checkpoint), with GRPO training running for 15 epochs per stage until convergence. All implementations were developed in PyTorch using libraries from the Hugging Face ecosystem, including `transformers`, `datasets`, `peft`, and `trl`. We comprehensively evaluated fine-tuned SLM performance through quantitative textual metrics, qualitative reward model assessments, system-level efficiency benchmarks, and engineering process simulations. For quantitative analysis, we employed standard NLP metrics—BLEU, ROUGE (1, 2, L), METEOR, SacreBLEU, BERTScore, and cosine similarity based on Sentence Transformer embeddings [39]—to assess textual fidelity and overlap. Qualitative evaluation examined correctness, coherence, helpfulness, complexity, and verbosity using the Nvidia Nemotron-4-340B reward model, which provided scores on a 0–4 scale. We investigated the trade-off between model compression and predictive fidelity through structural pruning techniques. Both width-level (neuron-level) and depth-level (layer-level) pruning were guided by importance heuristics computed during fine-tuning, enabling systematic parameter reduction while monitoring downstream performance impact. To improve inference-time reliability—particularly for factual accuracy and reasoning consistency—we implemented a test-time scaling mechanism. This approach combines multi-path exploration via stochastic sampling, confidence-based candidate ranking, iterative self-reflection and revision, and consensus aggregation. Together, these techniques enhance output robustness compared to standard deterministic decoding, as measured by qualitative reward model metrics. We conducted a systematic evaluation of optimization techniques to enhance performance (speed, memory usage, throughput) during autoregressive inference of fine-tuned SLMs, including the LLaMA-3-70B architecture on NVIDIA H100 GPUs. Our analysis focused on inference-time methods: PagedAttention [24] with KV-cache quantization for memory efficiency, Lookahead Decoding [12] for throughput improvement, and FlashAttention [8, 9] for latency reduction. These techniques were benchmarked using metrics including inference throughput (tokens/sec), average generation latency (sec), maximum batch size, and peak GPU

memory usage (GB), demonstrating their complementary benefits for demanding engineering applications. PagedAttention addressed memory fragmentation and throughput limitations by organizing the Key-Value cache into non-contiguous memory blocks, improving memory efficiency and enabling larger batch sizes compared to traditional contiguous caching. Lookahead Decoding accelerated autoregressive generation by reducing end-to-end latency through parallel token generation and verification within each forward pass while maintaining output equivalence with greedy decoding. We quantified its effectiveness through comparative measurements of generation latency and throughput against baseline methods. Finally, FlashAttention optimized attention computation by alleviating memory bandwidth bottlenecks inherent to Transformer architectures. As an I/O-aware solution, it improved attention mechanism efficiency, with its impact evaluated on both training/inference throughput and peak memory consumption during training. Finally, engineering feasibility was validated using the DWSIM open-source chemical process simulator [32]. Textual PFD (Process Flow Diagram) descriptions generated by the framework were manually translated into DWSIM flowsheets to verify material/energy balances and thermodynamic consistency through simulation, ensuring the results were chemically and physically valid—not merely syntactically correct flowsheets. Control strategies derived from textual PID (Piping and Instrumentation Diagram) descriptions were functionally assessed by implementing control loops in DWSIM's dynamic simulation environment, evaluating closed-loop stability and performance metrics including setpoint tracking and disturbance rejection.

3.3 Results

We present a comparative evaluation of Llama-3.2-1B and SmolLM-135M across successive fine-tuning stages on the respective held-out test splits of SFT datasets (Factual QA, SynDIP, and LogiCore), preference alignment fine-tuning datasets (DPO), and retrieval-augmented fine-tuning (RAFT) datasets (Local/Global RAIT), and report their performance against a comprehensive set of evaluation metrics. As shown in Figures 4a–4f, the evaluation was conducted using both token-level n-gram overlap metrics (BLEU, ROUGE-1/2/L, METEOR, SacreBLEU) and embedding-based semantic similarity metrics (BERTScore and Sentence-BERT cosine similarity), with all scores normalized to the [0,1] interval. We report the Llama-3.2-1B (see Figure 4a) performance on the test splits of the Factual QA, SynDIP, and LogiCore datasets. It demonstrates strong semantic alignment, evidenced by a high BERTScore and sentence similarity, despite lower performance on n-gram metrics, indicating a preference for paraphrastic generation over lexical overlap. In contrast, SmolLM-135M (see Figure 4d) performance on the test splits exhibits relatively higher n-gram scores and sentence similarity, while achieving moderate BERTScore, suggesting a tendency toward surface-level fidelity. We now report the performance of Llama-3.2-1B on the test split of the DPO dataset (refer Figure 4b) shows high semantic similarity profiles, whereas SmolLM-135M (Figure 4e) shows balanced improvements across both lexical and semantic metrics, reflecting effective alignment via instruction tuning. We now evaluate the Llama-3.2-1B performance on the test splits of the Local and Global RAIT datasets (refer Figure 4c), which continue to show dominant semantic scores relative to n-gram metrics. SmolLM-135M (Figure 4f) exhibits comparatively lower scores across most metrics, with sentence similarity remaining the strongest, suggesting diminished generalization ability under retrieval-augmented long-context settings. These plots provide phase-by-phase performance diagnostics, highlighting how successive fine-tuning regimes induce distinct response behaviors across models in terms of semantic coherence, lexical fidelity, and alignment with training objectives. In addition, we conduct a systematic evaluation of how fine-tuning (FT) and Graph RAG affect qualitative performance across six language model variants, comprising two model architectures at different scales: the larger Llama-3.2-1B and more compact SmoLLM2-135M. Each model variant represents a distinct configuration: (a) Llama-3.2-1B with both FT and Graph RAG (W/FT W/Graph RAG), (b) Llama-3.2-1B with FT but without Graph RAG (W/FT W/o Graph RAG), (c) Llama-3.2-1B without either FT or Graph RAG (W/o FT W/o Graph RAG), (d) Llama-3.2-1B without FT but with Graph RAG (W/o FT W/Graph RAG), (e) SmoLLM2-135M with both FT and Graph RAG (W/FT W/Graph RAG), and (f) SmoLLM2-135M with FT but without Graph RAG (W/FT W/o Graph RAG). The variants performance is rigorously assessed using the NVIDIA Nemotron-4-340B reward model across five key qualitative dimensions: helpfulness (measuring practical utility), correctness (factual accuracy), coherence (logical flow), complexity (depth of content), and verbosity (response length), with detailed results presented in Figures 5a-e. The evaluation reveals several important findings regarding model scale and methodological impact. Among Llama-3.2-1B variants, the FT+RAG configuration (variant a) demonstrates superior performance, achieving the highest scores in correctness and complexity by effectively combining fine-tuned capabilities with retrieved knowledge, though this comes with increased verbosity due to the incorporation of supplementary content from the knowledge graph. The FT-only variant (b) maintains strong performance in coherence and helpfulness but shows limitations in knowledge-intensive tasks without retrieval support. Notably, even without fine-tuning, the Graph RAG-enabled Llama variant (d) outperforms the baseline (c) in correctness, demonstrating that retrieval augmentation can partially compensate for the absence of task-specific tuning. However, the complete absence of both methods (variant c) results in the weakest performance, highlighting the limitations of relying solely on pretrained knowledge. For the smaller SmoLLM2-135M models, Graph RAG improves correctness (variant e vs. f), but both configurations underperform relative to the corresponding Llama-3.2-1B variants across all metrics, notably in coherence and complexity. This performance gap underscores the importance of model scale in effectively utilizing both fine-tuning and retrieval augmentation. The results

demonstrate that while FT substantially enhances overall response quality across all metrics by aligning the model with domain-specific requirements, Graph RAG provides complementary benefits primarily for factual accuracy. This combination proves particularly valuable in specialized domains like chemical process synthesis, where both task adaptation and external knowledge integration are crucial for high-quality generation. The study conclusively shows that the optimal configuration-Llama-3.2-1B with both FT and Graph RAG-achieves the most balanced performance across all evaluation dimensions, successfully integrating structured knowledge retrieval with fine-tuned language understanding capabilities while maintaining reasonable verbosity levels. These findings have important implications for deploying language models in technical domains where both factual precision and contextual understanding are paramount.

4 Conclusion

Automating the generation of industrially viable Process Flow Diagrams (PFDs) and Piping and Instrumentation Diagrams (PIDs) is critical for accelerating chemical process scale-up. Our closed-loop framework achieves this by integrating domain-adapted small language models (SLMs) with physics-aware validation to enable end-to-end automation. The approach combines multi-stage SLM fine-tuning—leveraging synthetic datasets and retrieval augmentation from a hierarchical chemical knowledge graph—with rigorous simulation-based validation using DWSIM. Results demonstrate that the synergy between fine-tuning and Graph Retrieval-Augmented Generation (RAG) enables high-fidelity PFD/PID generation with strong generalization capabilities. Specifically, the framework exhibits robust performance in zero-shot synthesis of novel chemical production processes and excels at core engineering QA tasks, including PFD/PID interpretation and analysis. By unifying generative AI with first-principles engineering constraints, the framework effectively bridges the gap between digital discovery and industrial deployment, addressing key R&D bottlenecks. Future work will focus on expanding capabilities through deeper integration with closed-loop simulation for automated process optimization and improving usability through direct generation of standard-format engineering diagrams (e.g., Visio, CAD). This work presents a validated pathway toward more efficient and reliable AI-driven chemical process design.

References

[1] Vincent Abbott and Gioele Zardini. 2024. FlashAttention on a Napkin: A Diagrammatic Approach to Deep Learning IO-Awareness. *arXiv preprint arXiv:2412.03317* (2024).

[2] Achmad Anggaviya Alimin, Dominik P. Goldstein, Lukas Schulze Balhorn, and Artur M. Schweidtmann. 2025. Talking like Piping and Instrumentation Diagrams (P&IDs). *arXiv preprint arXiv:2502.18928* (2025).

[3] Vidhisha Balachandran, Jingya Chen, Lingjiao Chen, Shivam Garg, Neel Joshi, Yash Lara, John Langford, Besmira Nushi, Vibhav Vineet, Yue Wu, et al. 2025. Inference-Time Scaling for Complex Tasks: Where We Stand and What Lies Ahead. *arXiv preprint arXiv:2504.00294* (2025).

[4] Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. 2024. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. *arXiv preprint arXiv:2412.09078* (2024).

[5] Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, and Sercan Ö Arık. 2025. SETS: Leveraging Self-Verification and Self-Correction for Improved Test-Time Scaling. *arXiv preprint arXiv:2501.19306* (2025).

[6] Shimao Chen, Zirui Liu, Zhiying Wu, Ce Zheng, Peizhuang Cong, Zihan Jiang, Yuhan Wu, Lei Su, and Tong Yang. 2024. INT-FlashAttention: Enabling Flash Attention for INT8 Quantization. *arXiv preprint arXiv:2409.16997* (2024).

[7] Yuan Chiang, Elvis Hsieh, Chia-Hong Chou, and Janosh Riebesell. 2024. LLAMP: Large language model made powerful for high-fidelity materials knowledge retrieval and distillation. *arXiv preprint arXiv:2401.17244* (2024).

[8] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[10] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems* 36 (2023), 10088–10115.

[11] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[12] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057* (2024).

[13] Yuan Gao, Zujing Liu, Weizhong Zhang, Bo Du, and Gui-Song Xia. 2024. Bypass Back-propagation: Optimization-based Structural Pruning for Large Language Models via Policy Gradient. *arXiv preprint arXiv:2406.10576* (2024).

[14] Shreesha Gowiakar, Srinivasan Iyengar, Sameer Segal, and Shivkumar Kalyanaraman. 2024. An Agentic Approach to Automatic Creation of P&ID Diagrams from Natural Language Descriptions. *arXiv preprint arXiv:2412.12898* (2024).

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

[16] Jeff Guo and Philippe Schwaller. 2024. Saturn: Sample-efficient generative molecular design using memory manipulation. *arXiv preprint arXiv:2405.17066* (2024).

[17] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309* (2024).

[18] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems* 37 (2024), 132876–132907.

[19] Edwin Hirretier, Lukas Schulze Balhorn, and Artur M. Schweidtmann. 2022. Towards automatic generation of piping and instrumentation diagrams (P&IDs) with artificial intelligence. *arXiv preprint arXiv:2211.05583* (2022).

[20] Chenglong Kang, Xiaoyi Liu, and Fei Guo. [n. d.]. RetroInText: A Multimodal Large Language Model Enhanced Framework for Retrosynthetic Planning via In-Context Representation Learning. In *The Thirteenth International Conference on Learning Representations*.

[21] Aum Kendapadi, Kerem Zaman, Rakesh R Menon, and Shashank Srivastava. 2024. INTERACT: Enabling Interactive, Question-Driven Learning in Large Language Models. *arXiv preprint arXiv:2412.11388* (2024).

[22] Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened LLaMA: Depth Pruning for Large Language Models with Comparison of Retraining Methods. *arXiv preprint arXiv:2402.02834* (2024).

[23] Agustinus Kristiadi, Felix Strieth-Kalthoff, Marta Skreta, Pascal Poupart, Alán Aspuru-Guzik, and Geoff Pleiss. 2024. A sober look at LLMs for material discovery: Are they actually good for Bayesian optimization over molecules? *arXiv preprint arXiv:2402.05015* (2024).

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.

[25] Xinzhe Li. 2025. A Survey on LLM Test-Time Compute via Search: Tasks, LLM Profiling, Search Algorithms, and Relevant Frameworks. *arXiv*

preprint arXiv:2501.10069 (2025).

[26] Zhihang Lin, Mingbao Lin, Yuan Xie, and Rongrong Ji. 2025. CPPO: Accelerating the Training of Group Relative Policy Optimization-Based Reasoning Models. *arXiv preprint arXiv:2503.22342* (2025).

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

[28] Qin Liu, Wenxuan Zhou, Nan Xu, James Y Huang, Fei Wang, Sheng Zhang, Hoifung Poon, and Muhao Chen. 2025. Metascale: Test-time scaling with evolving meta-thoughts. *arXiv preprint arXiv:2503.13447* (2025).

[29] Yue Liu, Jiaying Wu, Yufei He, Hongcheng Gao, Hongyu Chen, Baolong Bi, Jiaheng Zhang, Zhiqi Huang, and Bryan Hooi. 2025. Efficient Inference for Large Reasoning Models: A Survey. *arXiv preprint arXiv:2503.23077* (2025).

[30] Haiquan Lu, Yefan Zhou, Shiwei Liu, Zhangyang Wang, Michael W Mahoney, and Yaoqing Yang. 2024. Alphapruning: Using heavy-tailed self regularization theory for improved layer-wise pruning of large language models. *Advances in Neural Information Processing Systems* 37 (2024), 9117–9152.

[31] Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. 2024. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv preprint arXiv:2405.04304* (2024).

[32] Daniel Medeiros. 2025. DWSIM: Open Source Process Simulator. https://dwsim.fossee.in Accessed April 15, 2025.

[33] OpenAI. 2024. text-embedding-3-small model. https://platform.openai.com/docs/guides/embeddings. Accessed: August 2024.

[34] Elton Pan, Soohyoung Kwon, Sulin Liu, Mingrou Xie, Yifei Duan, Thorben Prein, Killian Sheriff, Yuriy Roman, Manuel Moliner, Rafael Gómez-Bombarelli, et al. 2024. A chemically-guided generative diffusion model for materials synthesis planning. In *AI for Accelerated Materials Design–NeurIPS 2024*.

[35] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2024. vattention: Dynamic memory management for serving llms without pagedattention. *arXiv preprint arXiv:2405.04437* (2024).

[36] Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572* (2025).

[37] Ankit Singh Rawat, Veeranjaneyulu Sadhanala, Afshin Rostamizadeh, Ayan Chakrabarti, Wittawat Jitkrittum, Vladimir Feinberg, Seungyeon Kim, Hrayr Harutyunyan, Nikunj Saunshi, Zachary Nado, et al. 2024. A little help goes a long way: Efficient llm training by leveraging small lms. *arXiv preprint arXiv:2410.18779* (2024).

[38] Isaac Rehg. 2024. KV-Compress: Paged KV-Cache Compression with Variable Compression Rates per Attention Head. *arXiv preprint arXiv:2410.00161* (2024).

[39] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[40] Fabrizio Sandri, Elia Cunegatti, and Giovanni Iacca. 2025. 2SSP: A Two-Stage Framework for Structured Pruning of LLMs. *arXiv preprint arXiv:2501.17771* (2025).

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[42] Lukas Schulze Balhorn, Edwin Hirretier, Lynn Luderer, and Artur M Schweidtmann. 2023. Data augmentation for machine learning of chemical process flowsheets. *arXiv e-prints* (2023), arXiv–2302.

[43] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems* 37 (2024), 68658–68685.

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

[45] Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. 2025. When To Solve, When To Verify: Compute-Optimal Problem Solving and Generative Verification for LLM Reasoning. *arXiv preprint arXiv:2504.01005* (2025).

[46] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* (2024).

[47] Henry W Sprueill, Carl Edwards, Khushbu Agarwal, Mariefel V Olarte, Udishnu Sanyal, Conrad Johnston, Hongbin Liu, Heng Ji, and Sutanay Choudhury. 2024. ChemReasoner: Heuristic search over a large language model's knowledge space using quantum-chemical feedback. *arXiv preprint arXiv:2402.10980* (2024).

[48] Sakhinana Sagar Srinivas, Akash Das, Shivam Gupta, and Venkataramana Runkana. 2024. Accelerating Manufacturing Scale-Up from Material Discovery Using Agentic Web Navigation and Retrieval-Augmented AI for Process Engineering Schematics Design. *arXiv preprint arXiv:2412.05937* (2024).

[49] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).

[50] Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. 2025. The Curse of Depth in Large Language Models. *arXiv preprint arXiv:2502.05795* (2025).

[51] Shengkun Tang, Oliver Sieberling, Eldar Kurtic, Zhiqiang Shen, and Dan Alistarh. 2025. DarwinLM: Evolutionary Structured Pruning of Large Language Models. *arXiv preprint arXiv:2502.07780* (2025).

[52] Yijun Tian, Yikun Han, Xiusi Chen, Wei Wang, and Nitesh V Chawla. 2025. Beyond answers: Transferring reasoning capabilities to smaller llms using multi-teacher knowledge distillation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*. 251–260.

[53] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.

[54] Gabriel Vogel, Lukas Schulze Balhorn, and Artur M Schweidtmann. 2023. Learning from flowsheets: A generative transformer model for autocompletion of flowsheets. *Computers & Chemical Engineering* 171 (2023), 108162.

[55] Haorui Wang, Marta Skreta, Cher-Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yangqiao Zhu, et al. 2024. Efficient evolutionary search over chemical space with large language models. *arXiv preprint arXiv:2406.16976* (2024).

[56] Haihang Wu. 2024. LLM-BIP: Structured Pruning for Large Language Models with Block-Wise Forward Importance Propagation. *arXiv preprint arXiv:2412.06419* (2024).

[57] Junjie Yang, Junhao Song, Xudong Han, Ziqian Bi, Tianyang Wang, Chia Xin Liang, Xinyuan Song, Yichao Zhang, Qian Niu, Benji Peng, et al. 2025. Feature Alignment and Representation Transfer in Knowledge Distillation for Large Language Models. *arXiv preprint arXiv:2504.13825* (2025).

[58] Sherry Yang, Simon Batzner, Ruiqi Gao, Muratahan Aykol, Alexander Gaunt, Brendan C McMorrow, Danilo Jimenez Rezende, Dale Schuurmans, Igor Mordatch, and Ekin Dogus Cubuk. 2024. Generative hierarchical materials search. *Advances in Neural Information Processing Systems* 37 (2024), 38799–38819.

[59] Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. 2025. Towards thinking-optimal scaling of test-time compute for llm reasoning. *arXiv preprint arXiv:2502.18080* (2025).

[60] Zhaojian Yu, Yinghao Wu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. 2025. Z1: Efficient Test-time Scaling with Code. *arXiv preprint arXiv:2504.00810* (2025).

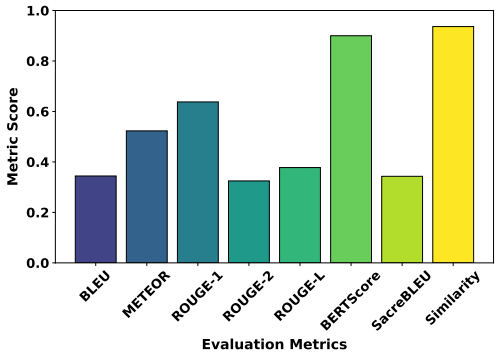
[61] Huan Zhang, Yu Song, Ziyu Hou, Santiago Miret, and Bang Liu. 2024. Honeycomb: A flexible llm-based agent system for materials science. *arXiv preprint arXiv:2409.00155* (2024).

[62] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. 2025. What, How, Where, and How Well? A Survey on Test-Time Scaling in Large Language Models. *arXiv preprint arXiv:2503.24235* (2025).

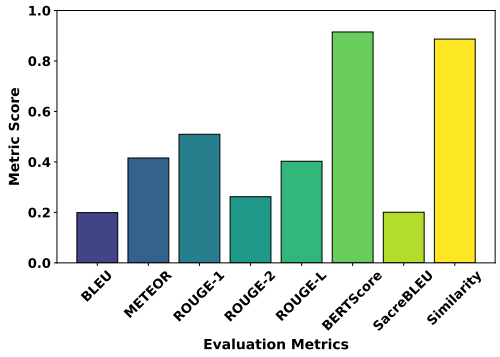
[63] Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. 2024. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6344–6355.

[64] Ming Zhong, Chenxin An, Weizhu Chen, Jiawei Han, and Pengcheng He. 2023. Seeking Neural Nuggets: Knowledge Transfer in Large Language Models from a Parametric Perspective. *arXiv preprint arXiv:2310.11451* (2023).

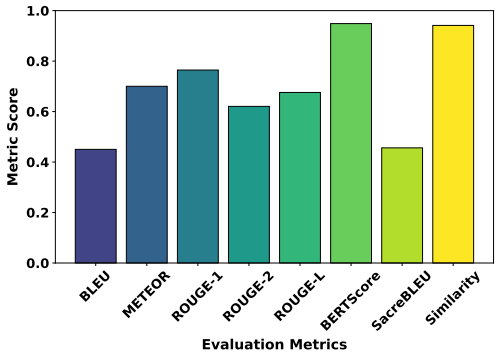
[65] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics* 12 (2024), 1556–1577.



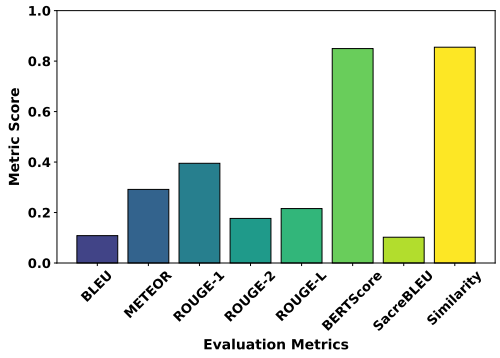
(a) Llama-3.2-1B evaluated on the test splits of the Factual QA, SynDIP, and LogiCore datasets, after Supervised Fine-Tuning (SFT) on the corresponding train splits. Metrics: BLEU, ROUGE, BERTScore, sentence similarity.



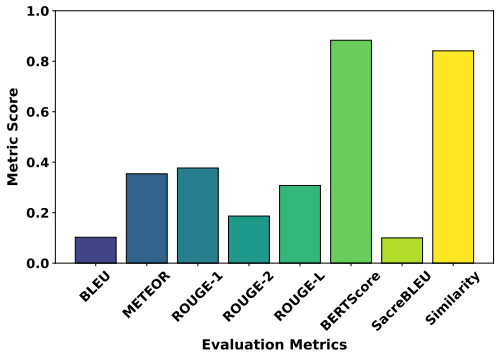
(b) Llama-3.2-1B evaluated on the DPO test split, after fine-tuning on the corresponding train set using Direct Preference Optimization (DPO). Metrics: BLEU, ROUGE, METEOR, SacreBLEU, BERTScore, sentence similarity.



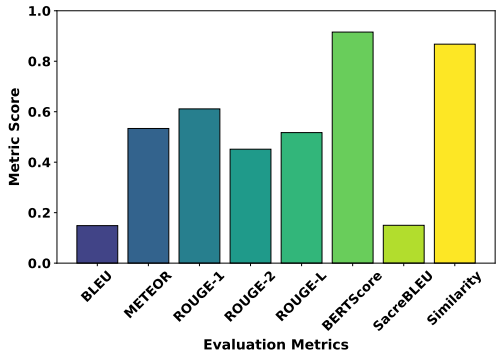
(c) Llama-3.2-1B evaluated on the test splits of the Local and Global RAIT datasets, after fine-tuning on the respective train splits using Retrieval-Augmented Instruction Tuning (RAIT). Metrics: BLEU, ROUGE, BERTScore, sentence similarity.



(d) SmolLM-135M evaluated on the test splits of the Factual QA, SynDIP, and LogiCore datasets, after Supervised Fine-Tuning (SFT) on the corresponding train splits. Metrics: BLEU, ROUGE, BERTScore, sentence similarity.

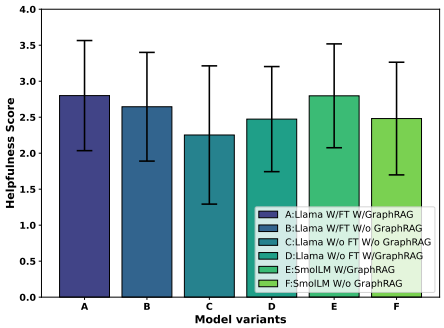


(e) SmolLM-135M evaluated on the DPO test split, after fine-tuning on the corresponding train split using Direct Preference Optimization (DPO). Metrics: BLEU, ROUGE, METEOR, SacreBLEU, BERTScore, sentence similarity.

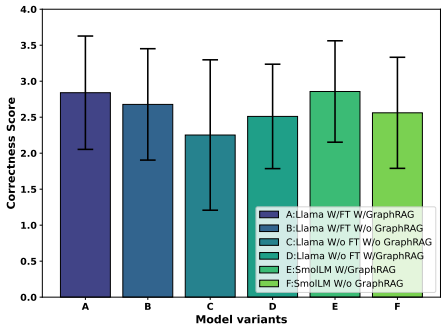


(f) SmolLM-135M evaluated on the test splits of the Local and Global RAIT datasets, after fine-tuning on the respective train splits using Retrieval-Augmented Instruction Tuning (RAIT). Metrics: BLEU, ROUGE, BERTScore, sentence similarity.

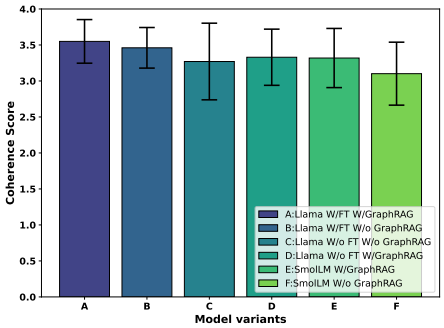
Figure 4: Quantitative evaluation of Llama-3.2-1B and SmolLM-135M across three fine-tuning stages: (1) Supervised Fine-Tuning (SFT) on the Factual QA, SynDIP, and LogiCore datasets; (2) Direct Preference Optimization (DPO) using the DPO dataset; and (3) Retrieval-Augmented Instruction Tuning (RAIT) on the Local and Global RAIT datasets. Performance is measured on held-out test splits corresponding to each training phase. Reported metrics span n-gram overlap (BLEU, ROUGE, METEOR, SacreBLEU) and semantic similarity (BERTScore, sentence similarity).



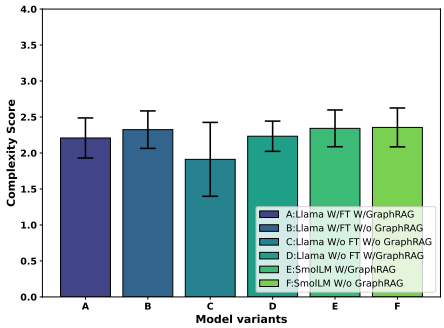
(a) Helpfulness. Fine-tuning and retrieval augmentation both improve helpfulness, with Llama-3.2-1B variants consistently outperforming SmolLM2-135M across configurations.



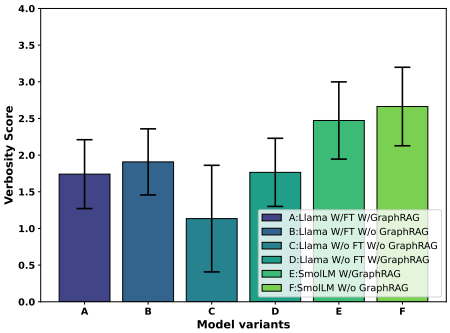
(b) Correctness. Graph RAG substantially enhances factual accuracy, particularly for Llama-3.2-1B, highlighting its role in mitigating hallucinations in knowledge-intensive tasks.



(c) Coherence. Fine-tuning leads to more logically structured outputs, with Llama-3.2-1B exhibiting stronger discourse consistency and narrative fluency.



(d) Complexity. Fine-tuned models produce more detailed and compositionally complex responses; retrieval augmentation further supports layered reasoning.



(e) Verbosity. Fine-tuning increases response length, while Graph RAG promotes more concise yet informative completions by constraining generation with retrieved context.

Figure 5: Qualitative evaluation of six model configurations on a 1.5K QA-pair out-of-distribution (OOD) benchmark, constructed independently of all synthetic training datasets (Factual QA, SynDIP, LogiCore, DPO, and RAIT). Models are scored by the Nvidia Nemotron-4-340B reward model along five qualitative dimensions: helpfulness, correctness, coherence, complexity, and verbosity. Results show that fine-tuning enhances structural quality, coherence, and content depth, while Graph RAG primarily contributes to factual precision. The Llama-3.2-1B model with both fine-tuning and retrieval augmentation achieves the highest scores across all dimensions, demonstrating the synergistic benefits of domain adaptation and structured knowledge retrieval for complex chemical process understanding.

5 Technical Appendix

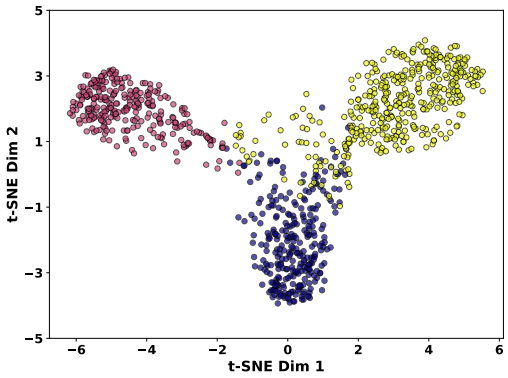


Figure 6: The figure shows a t-SNE visualization of embeddings generated by the OpenAI text-embedding-3-small model for the SynDIP dataset (containing PFD and PID descriptions) produced by GPT-4o using the ChemAtlas corpus. The plot reveals well-separated, compact clusters, indicating semantically similar groupings of process flow and instrumentation diagram descriptions across different chemicals. This demonstrates high inter-chemical consistency and strong intra-chemical structural relationships in the generated content.

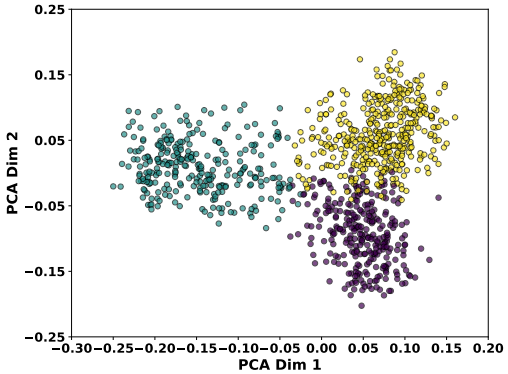


Figure 7: The figure shows a PCA plot of embeddings generated by the sentence embedding model for the process flow and instrumentation descriptions produced by GPT-4o using the ChemAtlas corpus. The first two principal components capture the maximum variance, revealing distinct patterns in the embedding space. The visualization shows tight clustering with well-defined groupings, reflecting high semantic consistency and strong domain alignment across process representations for different chemical production pathways.

5.1 Additional Results

We performed t-SNE and PCA visualizations to examine the clustering behavior of the process flow and instrumentation text embeddings using structured outputs from language models (such as GPT-4o and Claude Haiku) and unstructured agentic web-retrieved content from the ChemAtlas corpus. These projections assess inter-chemical consistency (semantic similarity of process flow and instrumentation descriptions across different but chemically related compounds) and intra-chemical coherence (semantic similarity among multiple process flow and instrumentation descriptions generated for the same chemical) by analyzing how semantically similar chemical production processes are grouped in embedding space. We illustrate these differences using embeddings computed from the OpenAI text-embedding-3-small model[33], which captures latent structure or similarity across chemical processes. For GPT-4o-generated outputs, Figures 6 and 7 show tight, well-separated clusters, reflecting strong semantic alignment across chemicals with similar synthesis pathways, equipment types, or control strategies. The descriptions of related chemical processes—such as those involving comparable unit operations or instrumentation—are embedded close together, while unrelated processes are clearly distinguished. In contrast, for Haiku-generated outputs, Figures 8 and 9 exhibit moderately compact clusters, indicating consistent grouping of chemically analogous processes with improved structural fidelity over web-derived data. Conversely, Figures 10 and 11 reveal diffuse and overlapping clusters for web-retrieved content, suggesting lower semantic regularity and weaker structural differentiation due to heterogeneous descriptions. In summary, the t-SNE and PCA plots for web-retrieved process flow and instrumentation descriptions show a mix of overlapping and distinct clusters, indicating partial inter-chemical consistency. While some chemical processes form coherent groupings, the overall dispersion reflects structural diversity and variation in semantic detail across uncurated web content. These clustering patterns facilitate few-shot prompting by identifying semantically similar chemical processes, enabling language models to transfer structural patterns—such as unit operation sequences, flow configurations, and control logic—from known processes to generate novel chemical production processes. This pattern recognition capability can be propagated through teacher-student transfer learning, where larger models first learn to identify and utilize these semantic clusters, then distill this knowledge to small-scale language models that can efficiently perform the same task. By retrieving industrial production processes for chemically similar neighbors within the same cluster, even these smaller models can generate accurate, contextually grounded process flow and instrumentation descriptions for previously unseen chemicals with minimal task-specific supervision. Overall, the PCA and t-SNE visualizations (see Figures 6, 7, 8, 9, 10, 11) demonstrate that structured synthetic generation exhibits tighter clustering and yields higher semantic consistency and clearer inter-chemical separation, whereas web-derived content exhibits noisier, less discriminative clustering. The Figures 12, 13, and 14 show the distribution of similarity scores between text embeddings of process flow and instrumentation descriptions generated from structured LLMs outputs and unstructured web-information retrieval. The highest alignment occurs between GPT-4o and Claude-3-Haiku (Figure 12), with a peak at 0.7–0.8, indicating strong semantic consistency in representing chemical processes. GPT-4o also aligns with web-retrieved data (Figure 13),

though the peak appears at a lower similarity range of 0.6–0.7, reflecting greater variation and reduced structural coherence. Haiku and web embeddings (Figure 14) exhibit a similar trend but with broader dispersion and weaker alignment. While web content exhibits partial semantic overlap, the tighter inter-model similarity highlights the coherence and reliability of LLM-generated descriptions.

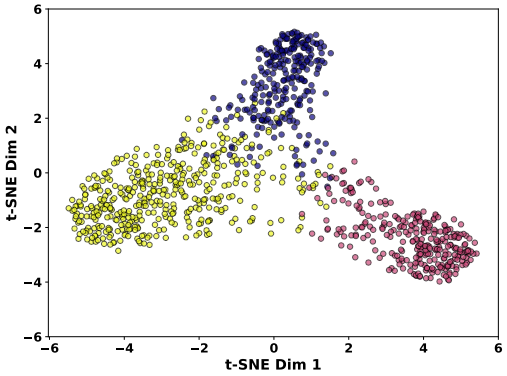


Figure 8: The figure shows a t-SNE visualization of text embeddings generated by the sentence embedding model for the process flow and instrumentation descriptions generated by Claude Haiku using the ChemAtlas corpus. The distinct clusters represent semantic relationships in the embedding space, showing moderate cluster separation. This suggests improved inter-chemical consistency and more stable intra-chemical representations compared to web-retrieved data.

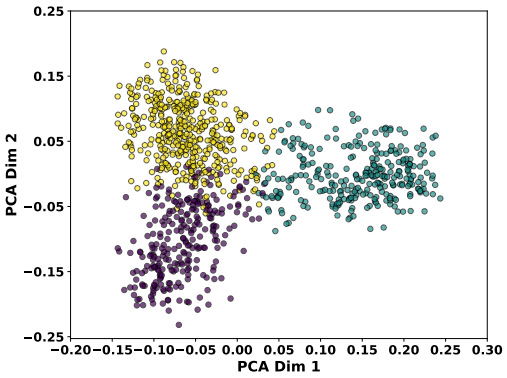


Figure 9: The figure shows a PCA visualization of text embeddings generated by the sentence embedding model for the process flow and instrumentation descriptions generated by Claude Haiku using the ChemAtlas corpus, displaying variance patterns across the first two principal components. The plot exhibits moderate clustering quality, indicating better structural consistency and improved grouping of production processes with similar chemical synthesis pathways compared to web-sourced data.

To summarize, Figures 12, 13, and 14 show that LLM-generated descriptions are more semantically aligned with each other

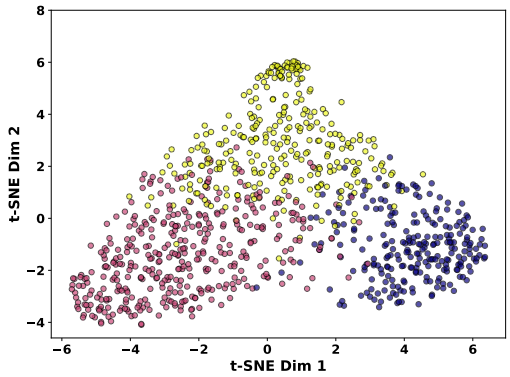


Figure 10: The figure shows a t-SNE visualization of text embeddings generated by the sentence embedding model for the process flow and instrumentation descriptions gathered from agentic web-retrieved data using the ChemAtlas corpus. The clusters show grouping patterns in two-dimensional space, displaying diffuse and overlapping formations. This indicates weaker inter-chemical consistency and lower intra-chemical structural coherence compared to LLM-generated synthetic data.

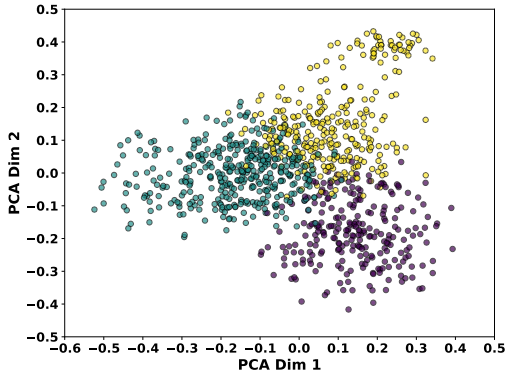


Figure 11: The figure shows a PCA visualization of text embeddings generated by the sentence embedding model for the process flow and instrumentation descriptions obtained from agentic web navigation using the ChemAtlas corpus, highlighting primary variance directions. The plot shows overlapping but loosely distributed embeddings across different chemicals, suggesting weaker structural coherence and less distinct grouping of similar chemical processes compared to synthetic data sources.

than with web content, indicating stronger inter-chemical consistency. The Figures 15–20 show training loss curves for the Llama 3.2 1B and SmolLM2-135M models fine-tuned with QLoRA on various synthetic datasets from the ChemAtlas corpus. For the Llama 3.2 1B model, supervised fine-tuning (SFT) on the Factual QA, SynDIP, and LogiCore datasets (Figure 15) reduces the loss from approximately 1.5 to 0.35 within 5 epochs (blue curve), with further improvement to around 0.1 after 15 epochs (red curve).

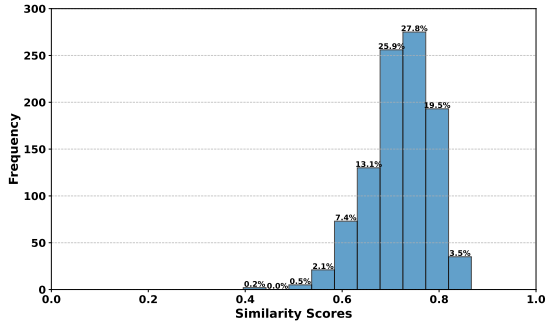


Figure 12: Cosine similarity distribution between process flow and instrumentation description embeddings generated by GPT-4o and Claude-3-Haiku, computed using a sentence embedding model. The prominent 0.7–0.8 mode reflects strong semantic agreement and high structural coherence in PFD–PID representations for id

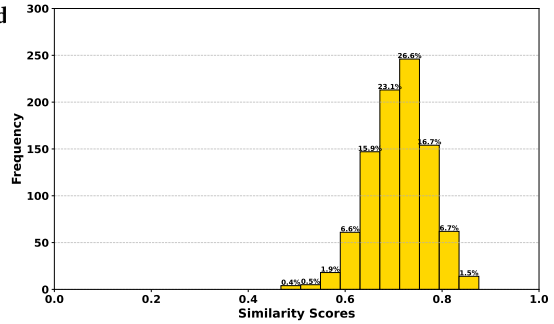


Figure 13: Cosine similarity distribution between GPT-4o-generated and web-retrieved process flow and instrumentation embeddings. The broader 0.6–0.7 peak, with higher variance, indicates moderate semantic alignment and greater representational variability compared to th

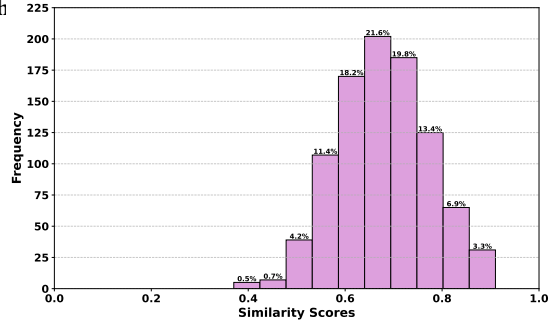


Figure 14: Cosine similarity distribution between Claude-3-Haiku and web-retrieved process flow and instrumentation description embeddings. The diffuse peak in the 0.6–0.7 range suggests weaker alignment and reduced structural consistency in representations relative to those generated by GPT-4o.

Direct Preference Optimization (DPO) training (Figure 16) demonstrates rapid convergence, achieving near-zero loss in the first epoch and maintaining stability throughout both 2-epoch and 5-epoch runs. When trained on multi-scale RAIT datasets (Figure 17), the model shows consistent improvement, with loss decreasing from roughly 0.15 to below 0.05 over 15 epochs. In contrast, the smaller SmolLM2-135M model exhibits slower and noisier convergence across all tasks. During SFT on the same QA datasets (Figure 18), the loss declines from approximately 2.2 to 0.6 but displays noticeable volatility throughout training. DPO fine-tuning (Figure 19) again leads to a sharp reduction in loss, reaching near-zero values within the first epoch. However, on the RAIT datasets (Figure 20), the model's loss decreases more gradually from around 1.5 to 0.2 over 15 epochs, with moderate fluctuations persisting during training.

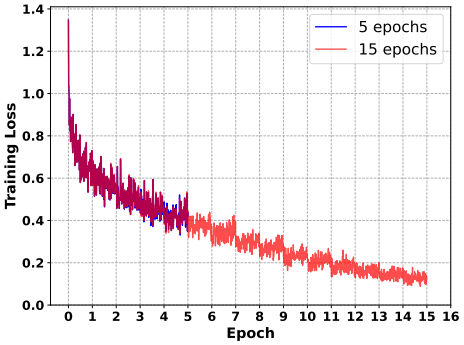


Figure 15: Supervised fine-tuning (SFT) loss for Llama 3.2 1B on Factual QA, SynDIP, and LogiCore datasets. Training loss decreases from ~1.4 to 0.35 in 5 epochs (blue) and reaches ~0.1 after 15 epochs (red), showing consistent convergence.

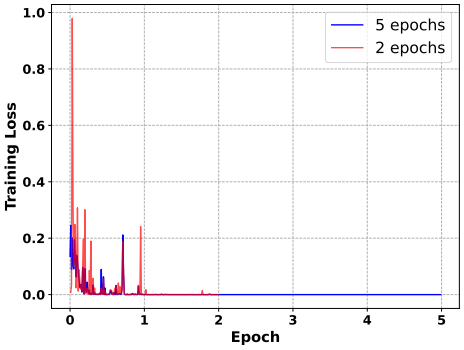


Figure 16: Direct Preference Optimization (DPO) loss for Llama 3.2 1B. The loss converges to near-zero within one epoch and maintains stability through both 2-epoch (red) and 5-epoch (blue) training runs.

These results demonstrate two key findings: (1) Llama 3.2 1B benefits significantly from extended training duration, and (2) SmolLM2-135M exhibits stronger dependence on the fine-tuning methodology, with DPO achieving more stable convergence than SFT. Figures 23a-23d quantify the computational resources required for fine-tuning both architectures, revealing that DPO consistently demands the fewest GPU hours while QA instruction tuning and Retrieval-Augmented Instruction Tuning (RAIT) show variable computational costs depending on dataset complexity.

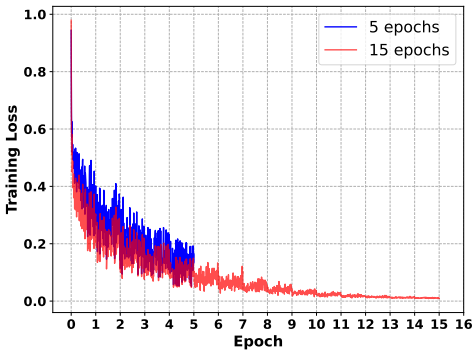


Figure 17: Training loss for Llama 3.2 1B on multi-scale RAIT datasets (Local/Global). The 5-epoch run (blue) achieves ~0.15 loss, while 15 epochs (red) reduce loss below 0.05, indicating effective learning.

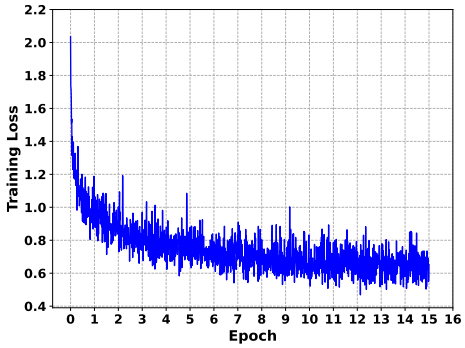


Figure 18: SFT loss for SmolLM2-135M on Factual QA, SynDIP, and LogiCore datasets. Loss decreases from ~2.1 to ~0.6 over 15 epochs, with higher variance than larger models.

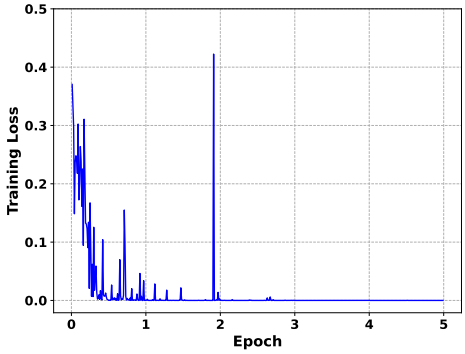


Figure 19: DPO loss for SmolLM2-135M. Initial loss of 0.35 reaches near-zero within one epoch and remains stable through 5 epochs, demonstrating efficient preference learning.

5.1.1 ***Beyond Training Data: Optimized Llama-3.2 1B vs. GPT-4o in Multi-Metric Evaluation on a Generalization Benchmark***. We conduct a comparative evaluation of the fine-tuned Llama-3.2 1B model (Llama FT) against GPT-4o using a 1.5K QA-pair generalization benchmark dataset, as shown in Figure 21. Performance is evaluated across five core metrics—helpfulness, correctness, coherence, complexity, and verbosity—each scored on a 0–4 scale using the Nvidia/Nemotron-4-340B reward model. This independent OOD benchmark is disjoint from the synthetic datasets used throughout model development—including both training and evaluation phases—which comprise Factual QA, SynDIP, LogiCore, and Local/Global RAIT and DPO. Each of these datasets includes predefined train, validation, and test splits. As shown in Figure 21, GPT-4o consistently achieves high scores across all metrics, establishing a strong performance baseline. The fine-tuned Llama-3.2 1B model demonstrates competitive results: it nearly matches GPT-4o in coherence, trails slightly in helpfulness and correctness, and produces significantly more concise responses (as indicated by lower verbosity scores). However, the larger error bars for Llama-3.2 1B suggest greater performance variability across samples. These findings show that despite its smaller size, Llama-3.2 1B rivals GPT-4o in key quality dimensions while offering practical benefits in response brevity and computational efficiency. We evaluate the zero-shot performance of the pretrained Llama-3.2 1B model augmented with GraphRAG and feedback mechanisms, without any additional fine-tuning on synthetic datasets. As shown in Figure 22, we evaluated three configurations on the same 1.5K QA-pair generalization benchmark dataset: (1) Llama-3.2 1B with both GraphRAG and feedback, (2) Llama-3.2 1B with GraphRAG but without feedback, and (3) Llama-3.2 1B without either GraphRAG or feedback. All configurations are evaluated using the Nvidia/Nemotron-4-340B reward model across the same five metrics. The results demonstrate that the configuration incorporating both GraphRAG and feedback consistently outperforms the other two variants, with especially significant gains in helpfulness and correctness—approaching a reward score of 3.0. These findings highlight the synergistic benefit of retrieval and critique mechanisms, even in the absence of task-specific fine-tuning, for enhancing zero-shot generalization. While coherence remained similar across configurations, the differences were less pronounced in this category.

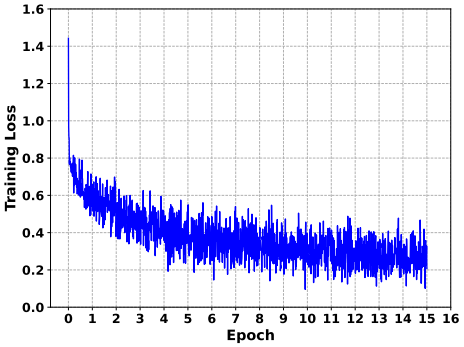


Figure 20: Training loss for SmolLM2-135M on RAIT datasets (Local/Global). Loss improves from ~1.4 to 0.2 over 15 epochs despite higher noise, showing gradual learning.

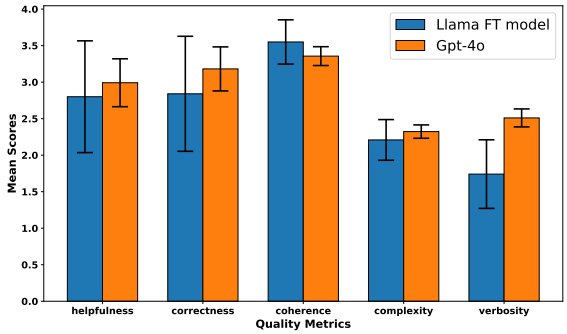


Figure 21: Comparative evaluation of the fine-tuned Llama-3.2 1B model and GPT-4o on a held-out 1.5K QA-pair generalization benchmark dataset, scored using the Nvidia/Nemotron-4-340B reward model. GPT-4o serves as a strong baseline due to its broad capabilities. The bar plot shows mean scores across five quality metrics—helpfulness, correctness, coherence, complexity, and verbosity—on a 0–4 scale. While GPT-4o slightly outperforms in most dimensions, the fine-tuned Llama-3.2 1B matches or exceeds it in coherence and exhibits significantly lower verbosity. Larger error bars for Llama-3.2 1B reflect greater variance across samples.

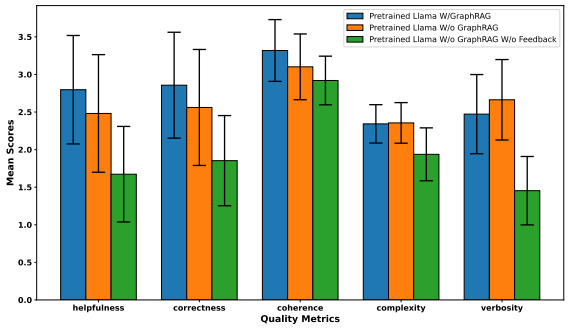
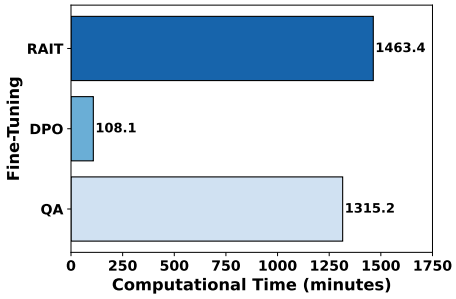


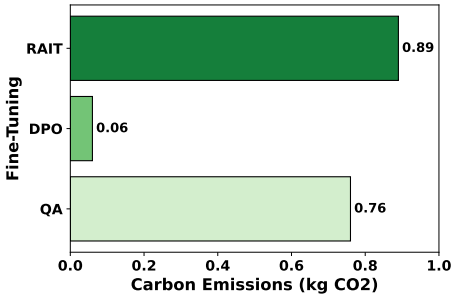
Figure 22: Zero-shot evaluation of the pretrained Llama-3.2 1B model under three configurations—(1) with GraphRAG and feedback, (2) with GraphRAG only, and (3) without GraphRAG or feedback—on the same 1.5K QA-pair generalization benchmark dataset. All configurations are scored using the Nvidia/Nemotron-4-340B reward model across five quality metrics: helpfulness, correctness, coherence, complexity, and verbosity (0–4 scale). Results show that combining GraphRAG with feedback yields the highest performance, while removing either component leads to consistent degradation, highlighting their complementary roles in enhancing zero-shot generalization.

These findings indicate that GraphRAG substantially enhances language model performance, especially in generating accurate and useful responses, while feedback mechanisms also contribute meaningful quality improvements even independently of GraphRAG.

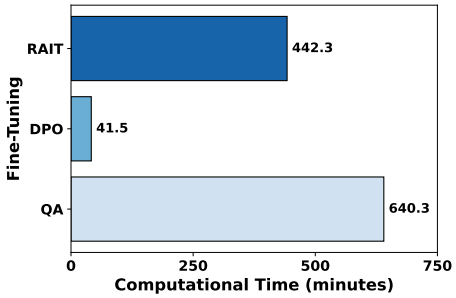
5.1.2 ***Ablation Study: Head-to-Head Multi-Metric Evaluation of Framework Variants.*** We evaluate six framework variants to analyze the individual and combined effects of fine-tuning (FT) and GraphRAG. Variant (A) Llama-3.2 1B w/FT w/GraphRAG represents the Llama-3.2 1B model with both fine-tuning and GraphRAG enabled. Variant (B) Llama-3.2 1B w/FT w/o GraphRAG uses the fine-tuned Llama-3.2 1B model but excludes GraphRAG. Variant (C) Llama-3.2 1B w/o FT w/GraphRAG employs the pre-trained Llama-3.2 1B model without fine-tuning but incorporates GraphRAG, while variant (D) Llama-3.2 1B w/o FT w/o GraphRAG serves as the baseline, featuring the pre-trained Llama-3.2 1B model with neither fine-tuning nor GraphRAG. For the smaller model, variant (E) SmolLM2-135M w/FT w/GraphRAG applies the fine-tuned SmolLM2-135M model with GraphRAG, and variant (F) SmolLM2-135M w/FT w/o GraphRAG represents the fine-tuned SmolLM2-135M model without GraphRAG. As shown in Figure 24, across all metrics - BERT (semantic similarity), BLEU (n-gram precision), METEOR (lexical and semantic alignment), and ROUGE (unigram, bigram, and longest-sequence overlap) - the results demonstrate that Variant A (Llama-3.2 1B with both fine-tuning and GraphRAG) consistently achieves the highest performance. Both fine-tuning and GraphRAG independently improve performance beyond the baseline, while their combination achieves peak performance. Specifically, Figure 24(a) presents BERT scores, which assess semantic similarity across the six framework variants. The results highlight the benefits of fine-tuning and GraphRAG: Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieves the highest score (0.9), indicating superior semantic alignment. Comparisons among the Llama-3.2 1B variants (A–D) show that fine-tuning and GraphRAG each independently improve performance over the baseline (D). A similar positive effect occurs for the fine-tuned SmolLM2-135M model, where GraphRAG enhances performance (E vs. F). These findings confirm that both methods improve semantic quality, with the fully configured Llama-3.2 1B model (Variant A) delivering the best performance. Figure 24(b) displays BLEU scores, measuring n-gram precision across the six framework variants. Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieves the highest score (0.17), outperforming other variants by a wide margin. Analysis of Llama-3.2 1B variants (A–D) shows that fine-tuning alone significantly improves precision over the pre-trained baseline (D), while the addition of GraphRAG further boosts performance (A vs. B). A comparable but smaller improvement occurs for the fine-tuned SmolLM2-135M model with GraphRAG (E vs. F). These results indicate that both fine-tuning and GraphRAG independently enhance precision, with their combined implementation in Variant A yielding optimal results. Figure 24(c) presents METEOR scores evaluating lexical and semantic alignment across the six variants, where fine-tuned Llama-3.2 1B models (Variants A/B, both >0.3) significantly outperform non-fine-tuned counterparts (Variants C/D). GraphRAG provides additional gains for both Llama-3.2 1B (A vs B, C vs D) and fine-tuned SmolLM2-135M (E vs F), confirming fine-tuning's primary role



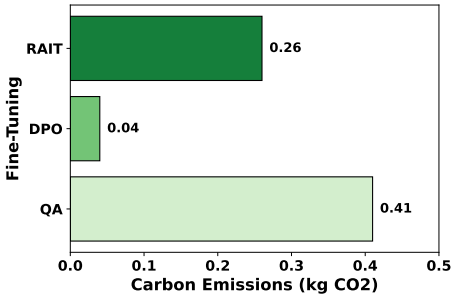
(a) Computational requirements for Llama 3.2 1B. RAIT demanded most wall-clock time (1463.4 min), followed by supervised QA tuning (1315.2 min), with DPO fastest (108.1 min).



(b) Environmental impact for Llama 3.2 1B. Supervised QA tuning produced highest CO_2 emissions (0.89 kg), followed by RAIT (0.76 kg), with DPO most efficient (0.06 kg).



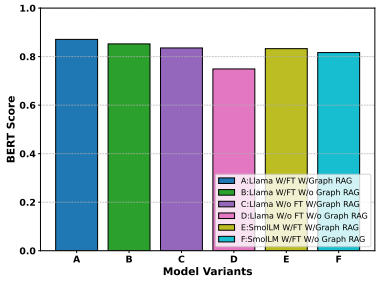
(c) Computational requirements for SmolLM2-135M. Supervised QA tuning required most time (640.3 min), followed by RAIT (442.3 min), with DPO fastest (41.5 min).



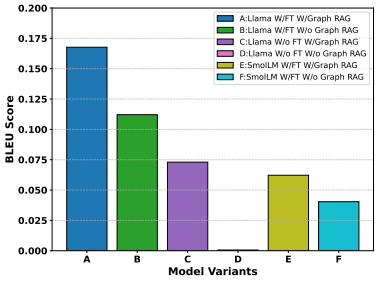
(d) Environmental impact for SmolLM2-135M. RAIT emitted most CO_2 (0.41 kg), followed by supervised QA tuning (0.26 kg), with DPO most efficient (0.04 kg).

Figure 23: Comparison of computational efficiency and environmental impact for fine-tuning Llama 3.2 1B (top) and SmolLM2-135M (bottom) across three approaches: (1) supervised QA tuning, (2) DPO, and (3) RAIT. Left panels (a,c) show wall-clock training time as a measure of computational requirements. Right panels (b,d) show resulting CO_2 emissions as a measure of environmental impact. DPO was consistently most efficient in both dimensions.

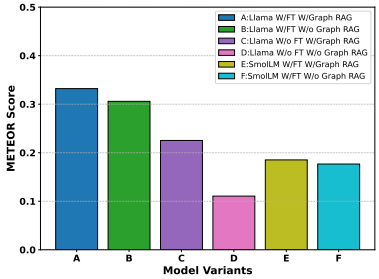
in score enhancement with GraphRAG offering secondary benefits - notably, top Llama configurations (A/B) consistently surpass all SmolLM2-135M variants. Figure 24(d) shows ROUGE-1 unigram overlap results, with Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieving the highest score (>0.5). Both fine-tuning and GraphRAG independently improve performance over the pre-trained baseline (Variant D), while GraphRAG also benefits the fine-tuned SmolLM2-135M (E vs F), demonstrating their synergistic effect on unigram overlap optimization with Variant A delivering peak performance. Figure 24(e) displays ROUGE-2 scores measuring bigram overlap across the six framework variants, where Variant A (Llama-3.2 1B with both fine-tuning and GraphRAG) achieves the highest score (>0.20). Both components independently enhance performance relative to the baseline (Variant D), with the fine-tuned SmolLM2-135M also showing GraphRAG benefits (E vs F). Figure 24(f) shows ROUGE-L scores evaluating sentence-level alignment, where Variant A again leads (0.26) - fine-tuning drives most improvement for Llama-3.2 1B (A vs B) while GraphRAG provides complementary gains, a pattern mirrored in SmolLM2-135M (E vs F). These results demonstrate that Variant A's combined approach yields optimal performance, with fine-tuning contributing primary improvements and GraphRAG offering secondary enhancements across both metrics. The consistent pattern across ROUGE-2 and ROUGE-L confirms the synergistic effect of these components in improving both bigram matching and longer-sequence alignment. Figure 25(a) reports SacreBLEU scores (n-gram precision) across six model variants, where Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieves superior performance ($\approx$ 0.168). Fine-tuning alone substantially boosts Llama-3.2 1B's scores (B vs D), with GraphRAG providing further enhancement (A vs B). The pretrained baseline (D) performs weakest, while GraphRAG also benefits the fine-tuned SmolLM2-135M (E vs F). These results establish that Variant A delivers optimal precision, with fine-tuned



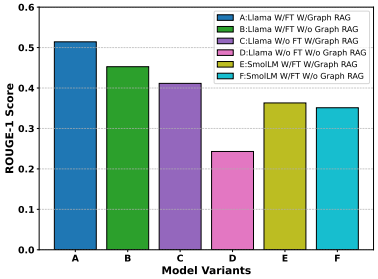
(a) BERTScore evaluation across six framework variants on the 1.5K QA-pair generalization benchmark. Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieves highest semantic similarity (~0.9), demonstrating the combined benefit of fine-tuning and GraphRAG.



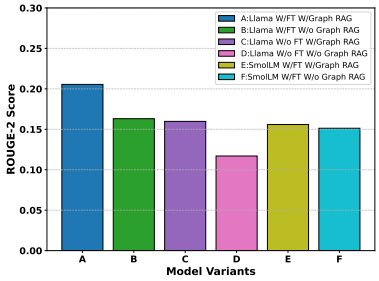
(b) BLEU score analysis showing n-gram precision improvements. Variant A (Llama-3.2 1B w/FT w/GraphRAG) leads with ~0.17 score, outperforming other configurations by significant margins.



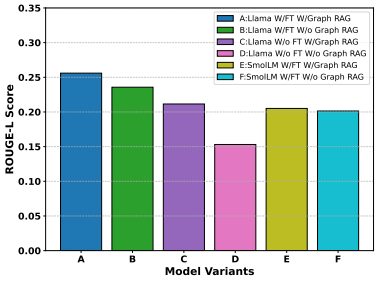
(c) METEOR scores assessing lexical/semantic alignment. Fine-tuned Llama-3.2 1B variants (A,B) score >0.3, with GraphRAG providing additional gains (A vs B).



(d) ROUGE-1 evaluation of unigram overlap. Variant A (Llama-3.2 1B w/FT w/GraphRAG) achieves >0.5 score, showing both fine-tuning and GraphRAG independently improve performance.

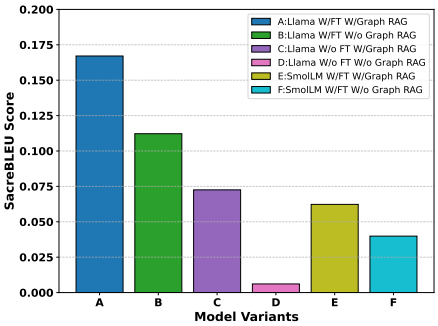


(e) ROUGE-2 analysis of bigram overlap. Variant A maintains lead (>0.20), with fine-tuned SmolLM2-135M also benefiting from GraphRAG (E vs F).

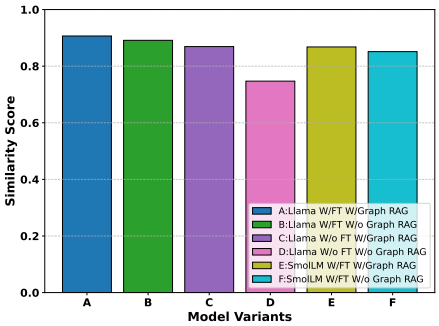


(f) ROUGE-L assessment of longest common subsequence. Variant A shows best performance (~0.26), with fine-tuning driving most improvement and GraphRAG providing complementary benefits.

Figure 24: Comprehensive evaluation of six framework variants (A–F) using standard NLP metrics on the 1.5K QA-pair generalization benchmark. Results demonstrate that Variant A consistently achieves top scores, with fine-tuning and GraphRAG offering complementary improvements. Configuration details: A=Llama-3.2 1B w/FT w/GraphRAG, B=Llama-3.2 1B w/FT w/o GraphRAG, C=Llama-3.2 1B w/o FT w/GraphRAG, D=baseline (Llama-3.2 1B w/o FT w/o GraphRAG), E=SmolLM2-135M w/FT w/GraphRAG, F=SmolLM2-135M w/FT w/o GraphRAG.

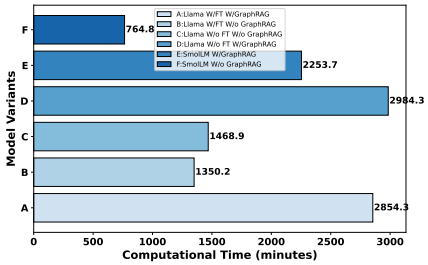


(a) SacreBLEU evaluation showing n-gram precision. Variant A (≈0.168) demonstrates optimal performance, with fine-tuning providing major gains and GraphRAG further enhancing results.

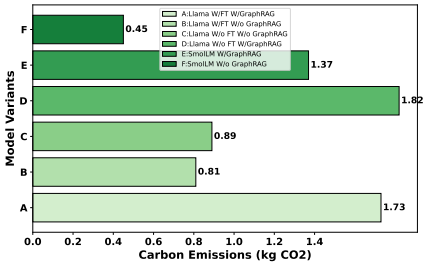


(b) Similarity score analysis reveals high semantic alignment (>0.85) across variants. Variant A peaks at ≈0.93, slightly exceeding Variant B and significantly outperforming smaller model configurations.

Figure 25: Additional metric evaluation (SacreBLEU, Similarity Score) for the six framework variants on the 1.5K QA-pair generalization benchmark. Results confirm the pattern observed in Figure 24: (1) fine-tuned Llama-3.2 1B variants (A,B) consistently outperform SmolLM2-135M counterparts (E,F), (2) the baseline configuration (D) remains weakest, and (3) Variant A (with both fine-tuning and GraphRAG) delivers optimal performance across all quality dimensions.



(a) Runtime (minutes) on the 1.5K QA-pair generalization benchmark. Llama-3.2 1B variants (A–D) require substantially longer inference time than SmolLM2-135M variants (E, F). For each model, enabling GraphRAG increases runtime (A vs B, D vs C, E vs F). Variant D is slowest; F is fastest.

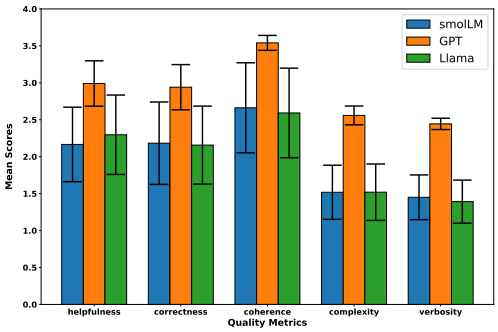


(b) Estimated CO_2 emissions during evaluation. Llama-3.2 1B variants (A–D) show higher emissions than SmolLM2-135M (E, F). GraphRAG consistently increases environmental cost (A vs B, D vs C, E vs F). Variant D yields the highest emissions; Variant F the lowest.

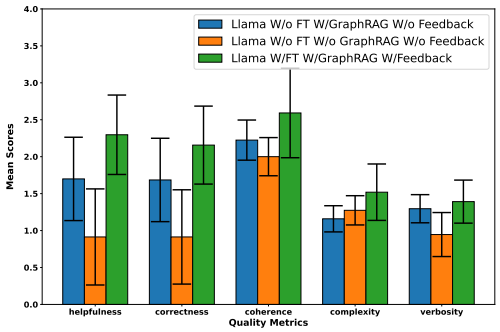
Figure 26: Evaluation-time computational cost and carbon impact across six framework variants (A–F) on the 1.5K QA-pair generalization benchmark. (a) Runtime in minutes. (b) Estimated CO_2 emissions (kg). SmolLM2-135M variants (E, F) are markedly more efficient. GraphRAG increases both runtime and emissions across all configurations.

Llama-3.2 1B consistently surpassing SmolLM2-135M counterparts. Figure 25(b) presents Similarity Scores measuring semantic alignment, where all variants score highly (> 0.85). Variant A peaks (≈ 0.93), slightly exceeding Variant B, demonstrating fine-tuning's major impact (B vs D) and GraphRAG's incremental value (A vs B). The pattern persists with Llama-3.2 1B (A,B) consistently outperforming fine-tuned SmolLM2-135M models (E,F), while baseline D remains weakest.

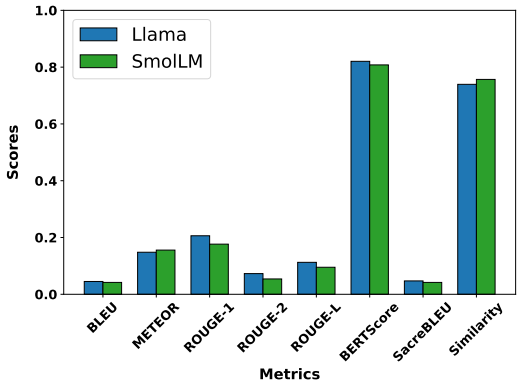
5.1.3 ***Computational Tradeoffs: Runtime and Carbon Costs Across Framework Variants.*** We now analyze the computational efficiency and environmental footprint of the six framework variants during evaluation on the 1.5K QA-pair generalization benchmark. Figure 26 quantifies both runtime and estimated CO_2 emissions across Llama-3.2 1B and SmolLM2-135M configurations. Larger Llama variants (A–D) consistently require more inference time (Figure 26a) and produce higher carbon emissions (Figure 26b) compared to their compact SmolLM counterparts (E, F). Within each model family, enabling GraphRAG leads to additional computational overhead and emissions—evident from comparisons such as A vs B, D vs C, and E vs F. Notably, Variant D (Llama-3.2 1B



(a) Model performance comparison on ChemEval database for PFD/PID generation, evaluated using the NVIDIA/Nemotron-4-340B reward model (0–4 scale). GPT-4o performs best across all metrics. Fine-tuned Llama-3.2 1B ranks second, with lower verbosity but higher variance. SmolLM2-135M scores lowest overall, though it matches Llama in complexity and verbosity.



(b) Performance of three Llama-3.2 1B configurations on ChemEval database. The fully enhanced variant (w/FT w/GraphRAG w/Feedback, green) outperforms the GraphRAG-only version (blue) and the base pretrained model (orange), showing the benefit of both retrieval and feedback.



(c) Comparison of fine-tuned Llama-3.2 1B and SmolLM2-135M on BLEU, METEOR, ROUGE, SacreBLEU, BERTScore, and Similarity for ChemEval database. Llama achieves higher overlap scores; both models perform well on semantic similarity.

Figure 27: Comprehensive evaluation of model performance on ChemEval database for automatic generation of PFD/PID descriptions. (a) Compares fine-tuned SmolLM2-135M, GPT-4o, and fine-tuned Llama-3.2 1B using reward model scores. (b) Analyzes the effects of fine-tuning, GraphRAG, and feedback on Llama-3.2 1B. (c) Benchmarks Llama-3.2 1B against SmolLM2-135M using standard quantitative metrics.

w/o FT w/GraphRAG) incurs the highest computational cost and carbon output, while Variant F (SmolLM2-135M w/FT w/o GraphRAG) emerges as the most resource-efficient configuration. These results highlight a clear tradeoff between model size, retrieval augmentation, and evaluation efficiency.

5.1.4 ***ChemEval Benchmark: Comparative Evaluation of Customized SLMs and GPT-4o.*** Figure 27 presents a comprehensive evaluation of model performance on the ChemEval benchmark for automatic PFD/PID generation, assessed using the NVIDIA/Nemotron-4-340B reward model and standard NLP metrics. We compare the proposed fine-tuned models—Llama-3.2 1B and SmolLM2-135M—with GPT-4o to assess generalization and generation quality in zero-shot chemical synthesis settings. Figure 27(a) reports mean reward model scores (scale: 0–4) across five qualitative dimensions: helpfulness, correctness, coherence, complexity, and verbosity. GPT-4o

establishes a strong upper bound across all metrics. The fine-tuned Llama-3.2 1B achieves second-best performance, outperforming SmolLM2-135M in helpfulness and coherence while producing more concise outputs (lower verbosity). However, it exhibits greater variance across samples, as indicated by wider error bars. The fine-tuned SmolLM2-135M achieves the lowest overall scores but performs comparably to Llama-3.2 1B in complexity and verbosity. Figure 27(b) examines the impact of architectural components within Llama-3.2 1B across three configurations: (1) a base pretrained model without fine-tuning, retrieval, or feedback; (2) the same model augmented with GraphRAG; and (3) the fully enhanced variant incorporating fine-tuning, GraphRAG, and feedback. The results demonstrate that both retrieval and feedback contribute independently to performance improvements, while their combination yields the strongest gains across all metrics. Figure 27(c) presents quantitative evaluation on BLEU, METEOR, ROUGE (1, 2, L), SacreBLEU, BERTScore, and cosine similarity. The Llama-3.2 1B model achieves higher scores on overlap-based metrics, while both models demonstrate strong alignment on semantic similarity tasks, confirming the ability of smaller LLMs to preserve semantic fidelity when appropriately fine-tuned.

5.2 Agentic Web Search for Automated Extraction and Synthesis of PFD/PID Descriptions in Chemical Processes

PFDs and PIDs are fundamental engineering schematics in the chemical process industry. They serve as the primary graphical representations of chemical plants. A PFD provides a high-level overview of a plant's major process units, primary piping, and the flow of materials and energy, outlining the transformation of raw materials into final products. In contrast, a PID offers a detailed schematic of the plant's mechanical components, including all piping, valves, instrumentation, and control systems required for safe and efficient operation and maintenance. We employ agentic web navigation—an advanced autonomous framework for web-based information retrieval—to collect and synthesize insights for generating PFD and PID descriptions of chemical processes from the ChemAtlas database. This framework builds foundational knowledge about established industrial processes used to manufacture common chemical products. Its outputs are detailed textual summaries comprising process descriptions, equipment specifications, and instrumentation details. At the core of the agentic web search framework is a meta-agent responsible for query decomposition, task delegation, and response integration. Given a complex input query Q, the meta-agent decomposes it into a set of subtasks $\{q_1, q_2, \ldots, q_n\}$, where each subtask represents a semantically coherent information need. For each subtask q_i, the meta-agent selects the optimal expert agent—such as the Visual Miner Agent, Research Agent, Patent Agent, or Wiki Agent—based on the highest semantic similarity between the vector representation of the subtask and that of the agent's capability. This approach goes beyond naïve task-to-tool mapping by embedding both task intent and agent capabilities into a shared semantic space, enabling principled and adaptive agent selection.

$$t_j^* = \arg\max_j \text{sim}_{\cos}(v(q_i), v(d_j))$$

where,

$$\text{sim}_{\cos}(v(q_i), v(d_j)) = \frac{v(q_i) \cdot v(d_j)}{\|v(q_i)\| \|v(d_j)\|}$$

Here, $v(q_i)$ and $v(d_j)$ denote the dense vector embeddings of the subtask and the expert agent's capabilities, respectively. The agent embedding $v(d_j)$ encodes domain expertise (i.e., specialized knowledge and skills relevant to retrieving and interpreting information within a specific content domain), tool access (e.g., SerpAPI), and reasoning modality (e.g., extractive or abstractive). Each expert agent operates within a multimodal, domain-specific retrieval regime. The Visual Miner Agent uses SerpAPI to retrieve high-quality industrial schematics—such as PFDs and PIDs—and parses them to generate semantic summaries using an LLM. The Research, Patent, and Wiki Agents also leverage SerpAPI to retrieve content from domain-specific corpora, including peer-reviewed scientific papers, technical reports, patents, and Wikipedia articles, respectively, and synthesize structured, contextual summaries using LLMs. Subtasks are then structured as nodes $V = \{v_1, \ldots, v_n\}$ in a Directed Acyclic Graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where edges $e_{ij} \in \mathcal{E}$ represent precedence constraints. This introduces formalisms into agent planning, moving away from fixed chain-of-thought paths to dynamic computation graphs. The DAG allows for topological sorting, task parallelism, and dependency resolution, supporting robust and interpretable execution flows. In particular, when no edge exists between subtasks q_i and q_j, their associated agents—such as the Visual Miner Agent, Research Agent, Patent Agent, or Wiki Agent—are executed in parallel to optimize latency and throughput. Each agent executes its assigned subtask q_i, retrieving a set of k candidate results $M = \{m_1, \ldots, m_k\}$, each scored using cosine similarity:

$$\text{sim}_{\cos}(v(m_i), v(q_i)) = \frac{v(m_i) \cdot v(q_i)}{\|v(m_i)\| \|v(q_i)\|}$$

The top-$K \leq k$ candidates are selected by ranking the retrieved items $m_i \in M$ in descending order of cosine similarity to $v(q_i)$, retaining the most relevant results for language model-based synthesis. Each expert agent then leverages a language model to perform information synthesis, semantic abstraction, and contextual reasoning over the selected top-ranked results, producing a coherent sub-answer R_{q_i}. The global answer A is constructed by integrating sub-answers:

$$A = \mathcal{F}_{\text{Meta}}(\{R_{q_i}\}_{i=1}^n)$$

To enhance quality and alignment, the framework introduces an iterative refinement loop for a predefined number of iterations:

$$A_{i+1} = \mathcal{F}_{\text{Meta}}(A_i, F_i)$$

Here, F_i includes feedback from: (a) LLM-as-Judge (e.g., GPT-4o, Anthropic Sonnet), applying ReAct-based reasoning and qualitative critique (e.g., correctness, coherence, and factuality); and (b) Reward Models (e.g., the Nemotron-4-340B

multidimensional reward model), which score candidate outputs based on five key attributes: helpfulness, correctness, coherence, complexity, and verbosity. These mechanisms form a self-correcting feedback loop, enabling reward-aligned output generation and enhancing factuality and task relevance. This modular, explainable framework extends RAG from static retrieval to agentic, feedback-driven generation of high-quality textual artifacts—enabling automated production of regulation-compliant PFD and PID descriptions for complex chemical synthesis pipelines. Figure 46 outlines our agentic framework for automated PFD/PID synthesis via query decomposition, expert routing, and iterative refinement.

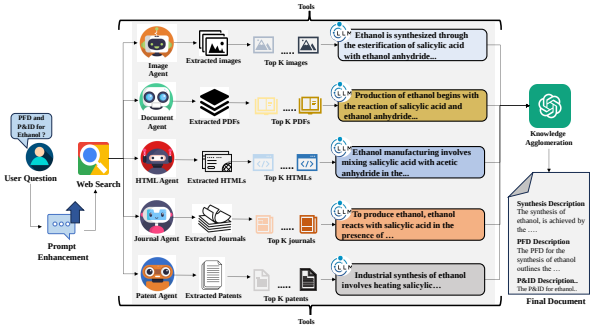


Figure 28: The figure illustrates an autonomous framework for generating textual descriptions of PFDs and PIDs for user-specified chemical processes to construct property graphs. A meta-agent decomposes complex queries into subtasks, routes them to domain-specific expert agents (e.g., Visual Miner, Research), and structures execution using a DAG. Agents retrieve multimodal content (e.g., PDFs, patents, HTMLs), rank results by relevance, and synthesize summaries using LLMs. Outputs are iteratively refined through LLM-as-Judge feedback and reward models to ensure accuracy and coherence.

5.3 Self-Instruct Synthetic Datasets for PFD/PID Interpretation: Reward-Model Validated LLM Distillation

We adopt a teacher–student transfer learning framework [21, 37, 52, 57, 64] that leverages large language models (LLMs), such as OpenAI's GPT-4o and Anthropic's Claude Haiku, as teacher models to generate high-quality synthetic training data. This synthetic dataset is then used to fine-tune smaller, open-source student models such as Llama 3.2 1B and SmolLM2-135M, enhancing their ability to follow complex instructions, provide helpful and context-aware responses, and perform specialized domain tasks—particularly the interpretation, analysis, and generation of PFDs and PIDs for chemical processes. From a Bayesian learning perspective, the teacher model approximates a posterior distribution over possible outputs, while the student model learns a compressed yet effective representation of this distribution. Through this knowledge distillation process, the student model achieves performance comparable to that of the teacher model on out-of-distribution (OOD) benchmarks, while being significantly more efficient to deploy. Our data generation pipeline employs self-instruct prompting, in which the teacher LLM is first conditioned on a small seed set of human-written instruction–response pairs, denoted as $\mathcal{D}_{\text{seed}} = \{(x_i, y_i)\}_{i=1}^{N}$, and then recursively generates synthetic pairs, represented by $\mathcal{D}_{\text{gen}} = \{(\tilde{x}_j, \tilde{y}_j)\}_{j=1}^{M}$, where $(\tilde{x}_j, \tilde{y}_j) \sim M(\cdot \mid \mathcal{D}_{\text{seed}})$. Here, $\tilde{x}_j$ is a synthetic instruction, $\tilde{y}_j$ is the corresponding generated response, and M denotes the teacher LLM. This bootstrapped approach produces instruction-style data without requiring extensive human annotation, forming the foundation of our dataset. In this section, we discuss the generation of multiple synthetic instruction–response datasets, all formatted as QA pairs, to support the development of specialized models for interpreting and generating PFD and PID descriptions in chemical process engineering. These datasets include: Factual QA, which targets domain-specific factual knowledge; SynDIP, designed to capture schematic-level descriptions of industrial processes; LogiCore, which elicits multi-step reasoning and procedural understanding; DPO, comprising chosen–rejected response pairs for preference optimization; and Local and Global RAIT, incorporating retrieval-augmented prompts with intra- and inter-cluster contextual grounding. All datasets are generated using a self-instruct bootstrapping pipeline with LLM-based prompting and validated through reward models to ensure alignment, informativeness, and response quality. (a) We generate a **factual QA dataset** (refer to Figure 32) by first selecting a domain-level topic $T \in \mathcal{T}$ (e.g., PFDs or PIDs), where $\mathcal{T}$ denotes the set of all possible topics. The teacher model M (e.g., GPT-4o) decomposes T into subtopics $\mathcal{S}_T = \{s_1, \ldots, s_n\}$, and then synthesizes question–answer pairs $(\tilde{q}_{jk}, \tilde{a}_{jk})$ for each subtopic s_j, where $j = 1, \ldots, n$ indexes the subtopics and $k = 1, \ldots, m_j$ indexes the QA pairs within subtopic s_j. Each pair is generated as:

$$(\tilde{q}_{jk}, \tilde{a}_{jk}) \sim M(\cdot \mid s_j, \mathcal{D}_{\text{seed}}^{\text{FQA}})$$

Here, $\mathcal{D}_{\text{seed}}^{\text{FQA}} = \{(x_i, y_i)\}_{i=1}^{N}$ denotes a seed set of human-written QA examples. The synthetic pairs form the dataset $\mathcal{D}_{\text{gen}}^{\text{FQA}} = \{(\tilde{q}_{jk}, \tilde{a}_{jk})\}_{j,k}$, which is filtered via a reward model (e.g., Nemotron-4-340B-Reward), defined as:

$$R(\tilde{q}, \tilde{a}) = \sum_{l=1}^{5} \alpha_l \cdot \text{Metric}_l(\tilde{q}, \tilde{a}),$$

where $\{\text{Metric}_l\}_{l=1}^{5} = \{H, C, Co, Cx, V\}$ represent helpfulness, correctness, coherence, complexity, and verbosity, respectively, and $\alpha_l \geq 0$ are predefined scalar weights. Only QA pairs satisfying the quality threshold $R(\tilde{q}, \tilde{a}) \geq \tau$ are retained, ensuring the dataset meets the quality standards required for downstream student model fine-tuning. The resulting dataset $\mathcal{D}_{\text{gen}}^{\text{FQA}}$ contains factual QA pairs related to PFDs and PIDs. (b) The **Direct Preference Optimization (DPO) dataset** (refer to Figure 33) is generated using the teacher model M (e.g., GPT-4o or Claude Haiku) and the reward model R. For each subtopic $s_j \in \mathcal{S}_T$ (derived from a domain-level topic T) and

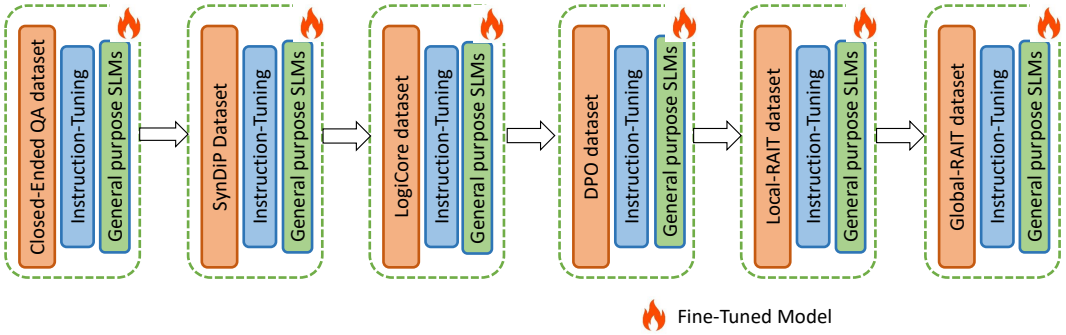


Figure 29: The figure illustrates the multi-stage instruction-tuning pipeline used to train specialized student models—such as Llama 3.2 1B and SmolLM2-135M—for PFD/PID interpretation. The pipeline integrates synthetic datasets—including Factual QA, SynDIP, LogiCore, DPO, Local-RAIT, and Global-RAIT—each generated using teacher LLMs (e.g., GPT-4o, Claude-3-Haiku) and validated with reward models such as Nvidia/Nemotron-340B. These datasets target diverse capabilities, including factual answering, process blueprint generation, logical reasoning, preference optimization, and retrieval-augmented comprehension. The combined instruction-tuning process refines general-purpose SLMs into fine-tuned models capable of performing domain-specific tasks in chemical process engineering.

each synthetic question $\tilde{q}_{jk} \in \mathcal{D}_{\text{gen}}^{\text{FQA}}$, we sample two candidate responses from M:

$$(\tilde{a}_{jk}^{+}, \tilde{a}_{jk}^{-}) \sim M(\cdot \mid \tilde{q}_{jk}, \mathcal{D}_{\text{seed}}^{\text{DPO}})$$

Here, $\tilde{a}_{jk}^{+}$ is the preferred response, $\tilde{a}_{jk}^{-}$ is the dispreferred alternative, and $\mathcal{D}_{\text{seed}}^{\text{DPO}}$ is a seed set of human-annotated preference pairs. The reward model R then computes the preference gap:

$$\Delta R_{jk} = R(\tilde{q}_{jk}, \tilde{a}_{jk}^{+}) - R(\tilde{q}_{jk}, \tilde{a}_{jk}^{-})$$

where R is defined as a weighted sum over five quality metrics: $\{H, C, Co, Cx, V\}$, representing helpfulness, correctness, coherence, complexity, and verbosity, respectively. Only preference triplets satisfying the quality threshold $\Delta R_{jk} \geq \tau_{\text{DPO}}$ are retained, forming the final dataset:

$$\mathcal{D}_{\text{gen}}^{\text{DPO}} = \{(\tilde{q}_{jk}, \tilde{a}_{jk}^{+}, \tilde{a}_{jk}^{-}) \mid \Delta R_{jk} \geq \tau_{\text{DPO}}\}$$

In summary, this pipeline automates the generation of high-quality preference-labeled datasets for PFD/PID tasks by combining teacher-model synthesis ($\tilde{a}_{jk}^{+}, \tilde{a}_{jk}^{-} \sim M$) with multi-metric reward-based filtering (R), resulting in DPO-optimized dataset tailored for domain-specific student model training. (c) The **SynDIP dataset** (refer to Figure 34) extends the teacher–student framework to generate structured chemical process context, PFDs, and PIDs textual descriptions, organized as sequential instruction–response pairs. The process context overview explains the why and how of a process design, covering its background, operation, engineering decisions, and control. It outlines unit operations, flow, reactions, and the rationale behind equipment and controls. For each target chemical, the teacher model M (e.g., GPT-4o or Claude-3-Haiku) generates a process blueprint $\tilde{b}_k$ in response to a fixed instruction template x_k^{SYN} (e.g., "Describe a chemical process for producing chemical X, including raw materials, reactions, and equipment"), with:

$$\tilde{b}_k \sim M(\cdot \mid x_k^{\text{SYN}}, \mathcal{D}_{\text{seed}}^{\text{SYN}})$$

where $\mathcal{D}_{\text{seed}}^{\text{SYN}}$ is a seed set of human-authored process blueprints. Each blueprint $\tilde{b}_k$ is then processed in two stages: (1) **PFD generation** via prompt x_k^{PFD} (e.g., "Convert this blueprint to a PFD: $[\tilde{b}_k]$"), yielding:

$$\tilde{f}_k \sim M(\cdot \mid x_k^{\text{PFD}}, \tilde{b}_k, \mathcal{D}_{\text{seed}}^{\text{PFD}})$$

where $\mathcal{D}_{\text{seed}}^{\text{PFD}}$ contains human-annotated PFD exemplars; and (2) **PID generation** using prompt x_k^{PID} (e.g., "Generate a PID for this PFD: $[\tilde{f}_k]$"), resulting in:

$$\tilde{p}_k \sim M(\cdot \mid x_k^{\text{PID}}, \tilde{f}_k, \mathcal{D}_{\text{seed}}^{\text{PID}})$$

where $\mathcal{D}_{\text{seed}}^{\text{PID}}$ contains human-annotated PID exemplars. The reward model R evaluates each instruction–response pair $(x_k, \tilde{y}_k)$—where $\tilde{y}_k \in \{\tilde{b}_k, \tilde{f}_k, \tilde{p}_k\}$—using the composite metric set $\{H, C, Co, Cx, V\}$ (helpfulness, correctness, coherence, complexity, verbosity). The final SynDIP dataset is defined as:

$$\mathcal{D}_{\text{gen}}^{\text{SynDIP}} = \{(x_k^{\text{SYN}}, \tilde{b}_k, x_k^{\text{PFD}}, \tilde{f}_k, x_k^{\text{PID}}, \tilde{p}_k) \mid R(x_k^{\text{SYN}}, \tilde{b}_k) + R(x_k^{\text{PFD}}, \tilde{f}_k) + R(x_k^{\text{PID}}, \tilde{p}_k) \geq \tau_{\text{SYN}}\}$$

ensuring that each entry includes validated process context, PFD, and PID descriptions for a complete chemical process representation. (d) The **LogiCore Dataset** (refer to Figure 35) extends our teacher-student framework to generate multi-step reasoning question-answer pairs for PFD/PID analysis. Using the same teacher model M (e.g., GPT-4o) and seed dataset $\mathcal{D}_{\text{seed}}^{\text{chem}}$ as in $\mathcal{D}_{\text{gen}}^{\text{FQA}}$, we synthesize logical QA pairs $(\tilde{q}_j, \tilde{a}_j) \sim M(\cdot \mid x_i, \mathcal{D}_{\text{seed}}^{\text{chem}})$, where each $\tilde{a}_j$ contains explicit chain-of-thought reasoning. These pairs are filtered via the established reward model R (Nemotron-4-340B-Reward) using the same metrics as in $\mathcal{D}_{\text{gen}}^{\text{FQA}}$: $R(\tilde{q}_j, \tilde{a}_j) = \sum_{l=1}^{4} \alpha_l \cdot \text{Metric}_l(\tilde{q}_j, \tilde{a}_j)$ $\{\text{Metric}_l\}_{l=1}^{4} = \{H, C, Co, Cx\}$ (helpfulness, correctness, coherence, complexity). The final dataset

$$\mathcal{D}_{\text{gen}}^{\text{LogiCore}} = \{(\tilde{q}_j, \tilde{a}_j) \mid R(\tilde{q}_j, \tilde{a}_j) \geq \tau_{\text{logic}}\}$$

retains only high-quality reasoning chains, with logical validity implicitly ensured through C (factual alignment with PFD/PID schematics) and Co (stepwise flow coherence), maintaining full consistency with our synthetic data generation framework. (e) The **Local RAIT Dataset** (refer to Figure 36) extends our teacher-student framework to retrieval-augmented generation. Unlike $\mathcal{D}_{\text{gen}}^{\text{FQA}}$ and $\mathcal{D}_{\text{gen}}^{\text{SynDIP}}$, Local RAIT integrates retrieval mechanisms to ground M's outputs in source documents, mitigating hallucination risks. For each chemical process description from the SynDip datasets for the ChemAtlas database (stored as PDF documents containing process flow and instrumentation descriptions), the raw text T is extracted and parsed into semantically coherent chunks $C_T = \{c_1, ..., c_K\}$, where $c_k \sim \text{Chunk}(T)$ and each c_k retains contextual continuity with neighboring chunks. The teacher model M (GPT-4o) then synthesizes QA pairs $(\tilde{q}_k, \tilde{a}_k) \sim M(\cdot \mid c_k, \mathcal{D}_{\text{seed}}^{\text{RAIT}})$, conditioned on seed human examples $\mathcal{D}_{\text{seed}}^{\text{RAIT}} = \{(x_i, y_i)\}$ that include both questions and gold-standard retrieval-augmented answers. This approach ensures $(\tilde{q}_k, \tilde{a}_k)$ are document-grounded, with c_k providing explicit provenance for generated answers—critical for technical domains where factual alignment with PFD/PID schematics is required.

$$\mathcal{D}_{\text{gen}}^{\text{LocalRAIT}} = \{(\tilde{q}_k, c_k, \tilde{a}_k) \mid R(\tilde{q}_k, \tilde{a}_k) \geq \tau \wedge \mathcal{L}(\tilde{q}_k, \tilde{a}_k) \geq 4\}$$

These pairs are filtered using the same reward model R and Likert scoring $\mathcal{L}$ as $\mathcal{D}_{\text{gen}}^{\text{FQA}}$, where $R(\tilde{q}_k, \tilde{a}_k) = \sum_{l=1}^{5} \alpha_l \cdot \text{Metric}_l(\tilde{q}_k, \tilde{a}_k)$ (metrics: H=Helpfulness, C=Correctness, Co=Coherence, Cx=Complexity, V=Verifiability against c_k) and $\mathcal{L}(\tilde{q}_k, \tilde{a}_k) \in \{1, ..., 5\}$ (1=Poor, 3=Average, 5=Excellent) evaluates answer quality across helpfulness, correctness, and coherence. Only examples meeting both thresholds (τ for R, 4+ for $\mathcal{L}$) are retained in $\mathcal{D}_{\text{gen}}^{\text{LocalRAIT}}$. (f) The **Global RAIT Dataset** (refer to Figure 37) scales retrieval-augmented generation to both intra- and cross-document comprehension. Chunks C_T are clustered into semantically related groups $\mathcal{G}_j$ via cosine similarity $\text{sim}(\phi(c_i), \phi(c_j)) \geq \gamma$, where ϕ is a domain-tuned embedding model (fine-tuned on $\mathcal{T}$ using contrastive learning) optimized for cross-document semantic relationships. For cross-document groups, $\mathcal{G}_j$ aggregates chunks from multiple source PDFs. The teacher model M generates multi-scale answers $(\tilde{a}_j^{\text{long}}, \tilde{a}_j^{\text{short}}) \sim M(\cdot|\mathcal{G}_j, \mathcal{D}_{\text{seed}}^{\text{Global}})$, conditioned on seed examples $\mathcal{D}_{\text{seed}}^{\text{Global}}$ that include cross-document QA pairs.

$$\mathcal{D}_{\text{gen}}^{\text{GlobalRAIT}} = \{(\tilde{q}_j, \mathcal{G}_j, \tilde{a}_j^{\text{long}}, \tilde{a}_j^{\text{short}}) \mid R(\tilde{a}_j^{\text{long}}) \geq \tau \wedge R(\tilde{a}_j^{\text{short}}) \geq \tau \wedge \mathcal{L} \geq 4\}$$

where $\tilde{a}_j^{\text{long}}$ provides multi-sentence, structured reasoning with evidence and trade-offs, while $\tilde{a}_j^{\text{short}}$ offers 1–2 sentence surface-level facts. Filtering follows the same criteria as $\mathcal{D}_{\text{gen}}^{\text{SynDIP}}$, applying both the reward threshold τ and a Likert score of $\mathcal{L} \geq 4$. By leveraging grouped chunk clusters, Global RAIT enables the student model to generate contextually grounded responses that synthesize information across intra- and inter-document contexts.

5.3.1 ***Computational Time Analysis for Synthetic Dataset Generation.*** Synthetic dataset generation follows a unified three-step pipeline: QA pair synthesis via teacher LLMs (e.g., GPT-4o, Claude Haiku), reward model validation (Nemotron-4-340B), and multi-metric filtering. SynDIP is the most time-intensive (2179.6 min) due to its sequential generation of process blueprints, PFDs, and PIDs. LogiCore (600.6 min) emphasizes multi-step reasoning; Global RAIT (480.4 min) involves cross-document clustering; and Local RAIT (320.7 min) targets chunk-level QA generation. DPO (201.8 min) and Factual QA (155.4 min) are faster, reflecting their simpler generation logic. This reflects a clear trade-off between dataset complexity and computational cost (Figure 30).

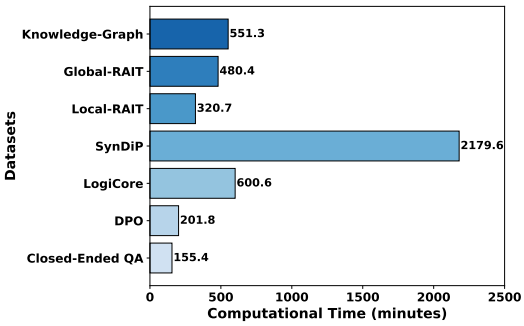


Figure 30: Computational time for generating self-instruct synthetic datasets, including QA pair creation, verification (using either the Nvidia Nemotron-4-340B reward model or an LLM-as-a-judge approach), and quality filtering. SynDiP's multi-stage generation (process context → PFD → PID) requires significantly more time than simpler factual QA generation due to its iterative refinement process

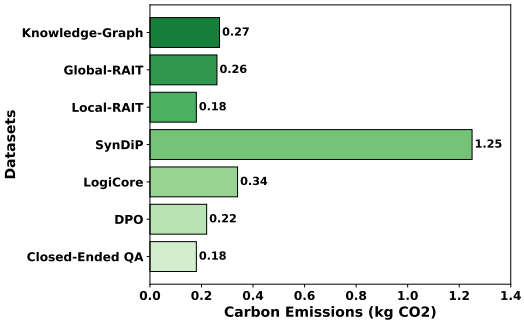


Figure 31: Carbon emissions (kg CO_2) for synthetic dataset generation. SynDIP incurs the highest emissions, while Factual QA, DPO, and Local RAIT exhibit the lowest.

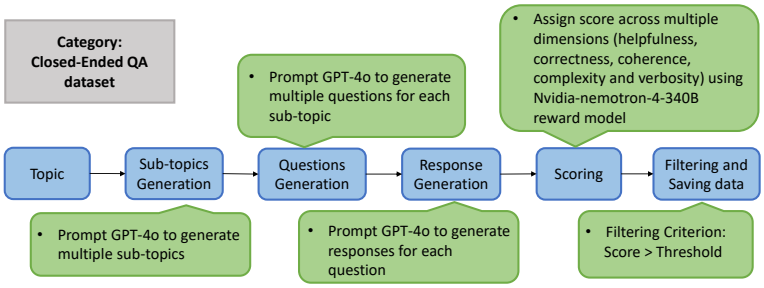


Figure 32: The figure shows the pipeline for generating synthetic Factual QA dataset. GPT-4o or claude-3-Haiku decomposes domain topics into subtopics and creates question-answer pairs, which are filtered by the Nvidia Nemotron-4-340B reward model based on metrics like correctness, coherence and etc. Only high-scoring pairs are retained for the final dataset.

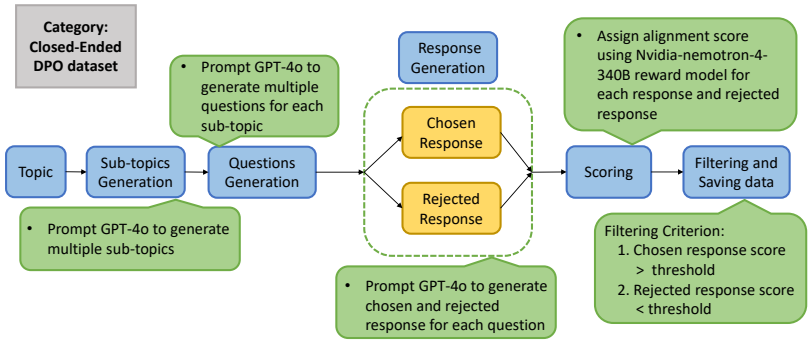


Figure 33: The figure illustrates the workflow for synthetic DPO dataset generation. GPT-4o or claude-3-Haiku generates questions with paired preferred and dispreferred responses. The Nvidia Nemotron-4-340B reward model scores responses, and pairs are filtered to ensure the preferred response ranks significantly higher in quality.

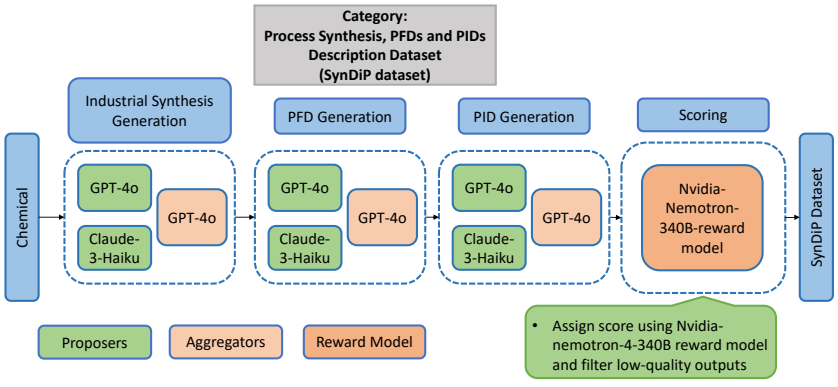


Figure 34: The figure outlines the SynDIP dataset generation process. Teacher models (GPT-4o, Claude-3-Haiku) first create overview of chemical process contexts, then derive PFD and PID descriptions. The Nvidia Nemotron-4-340B reward model validates each step, ensuring consistent quality across all outputs.

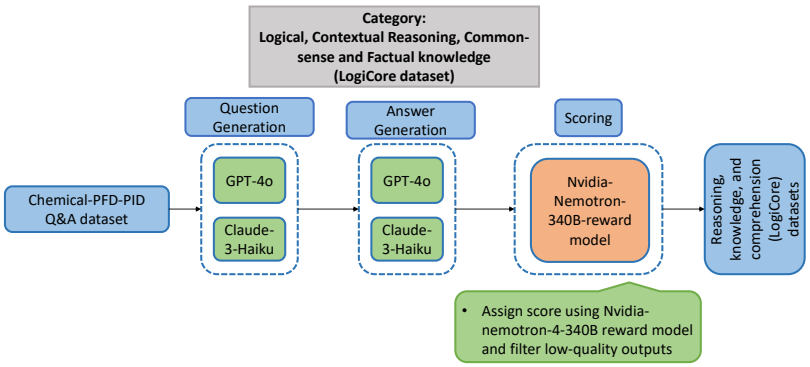


Figure 35: The figure outlines the generation pipeline for the LogiCore Dataset. Starting with a seed SynDIP dataset, GPT-4o and Claude-3-Haiku generate reasoning-augmented, Chain of thought (CoT) question-answer pairs. The Nvidia Nemotron-4-340B reward model scores outputs based on helpfulness, correctness, and coherence. High-quality responses are filtered to create a final dataset for advanced reasoning and contextual comprehension in process engineering.

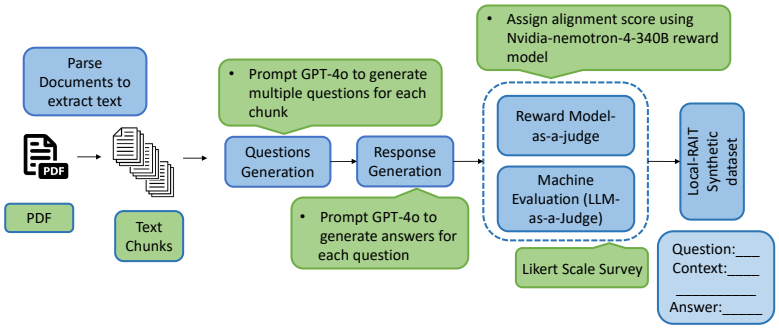


Figure 36: The figure depicts the workflow for the Local RAIT Dataset generation. Text chunks are extracted from the seed SynDIP dataset, and GPT-4o generates retrieval-grounded question-answer pairs. Outputs are evaluated using the Nemotron-4-340B reward model and additional LLM-based validation. High-quality pairs are retained to build a dataset for retrieval augmented instruction-tuning in process engineering tasks.

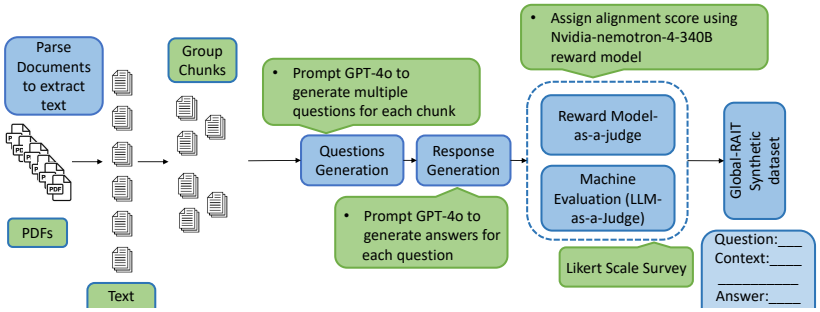


Figure 37: The figure illustrates the Global RAIT dataset generation workflow. PDFs text is parsed, chunked, and grouped via semantic clustering to preserve context. GPT-4o generates questions and multi-turn refined answers. A retriever selects top-k relevant chunks using vector similarity of embeddings obtained from a sentence embedding model. Outputs are validated through the Nvidia Nemotron-4-340B reward model, LLM-based checks, and Likert-scale feedback, yielding a high-quality dataset for instruction-tuning.

5.3.2 ***Carbon Emissions for Synthetic Dataset Generation.*** Carbon emissions scale with computational intensity during synthetic dataset generation and are measured using CodeCarbon[1] by tracking CPU/GPU energy consumption and grid carbon intensity. These emissions reflect the efficiency of the teacher–student framework involving self-instruct prompting, reward-model validation, and multi-stage data generation. SynDIP incurs the highest carbon footprint at 1.25 kg CO_2, due to the sequential generation of process context descriptions, PFDs, and PIDs with iterative reward-based filtering. LogiCore and Global RAIT exhibit moderate emissions of 0.34 kg CO_2 and 0.26 kg CO_2, respectively, corresponding to their focus on multi-step reasoning and retrieval-augmented generation. The DPO, Local RAIT, and Factual QA datasets demonstrate higher computational efficiency, recording lower emissions of 0.22 kg CO_2, 0.18 kg CO_2, and 0.18 kg CO_2, respectively, by leveraging lighter generation workflows and more targeted reward-model checks. These results highlight the environmental impact of synthetic dataset generation and underscore the importance of optimizing computational pipelines to align data augmentation strategies with sustainability goals, as shown in Figure 31.

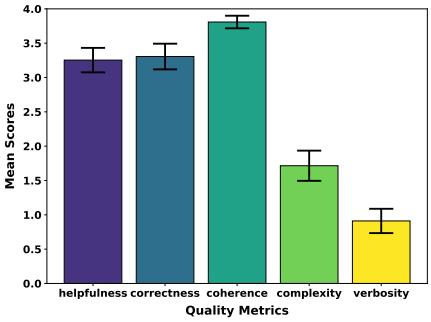


Figure 38: The figure shows the evaluation results for the generated Factual QA dataset using the NVIDIA-Nemotron-4-340B reward model. Each QA pair is scored on a 0–4 scale across five quality dimensions: helpfulness, correctness, coherence, complexity, and verbosity, ensuring high-quality data for instruction tuning.

5.3.3 ***Evaluation of Synthetic Datasets.*** Our teacher-student transfer learning framework utilizes large language models (LLMs) - with GPT-4o and Claude-3-Haiku for generation and the NVIDIA-Nemotron-4-340B reward model for evaluation - to create high-quality synthetic datasets for fine-tuning SLMs including Llama 3.2-1B, Qwen 2.5-1.5B, and SmolLM2-135M. These models are specifically optimized for domain-specific tasks involving PFD and PID analysis, interpretation, and generation. The approach enables precise output ranking and filtering that aligns with human preference criteria throughout the synthetic dataset creation and evaluation process. We rigorously assessed each dataset using the NVIDIA-Nemotron-4-340B reward model, which scores outputs on a 0-4 scale across five key metrics: helpfulness, correctness, coherence, complexity, and verbosity. Figure 38 presents the evaluation results for the Factual QA dataset, while Figures 39 and 40 show the performance comparison between chosen and rejected responses in the DPO dataset. For the multi-stage SynDIP dataset generating process context, PFDs and PIDs, Figure 41 demonstrates its quality, and Figure 42 evaluates the multi-step reasoning in the LogiCore dataset. Finally, the retrieval-augmented datasets are assessed in Figures 43 (Local RAIT) and 44 (Global RAIT), collectively confirming each dataset's quality and suitability for their specific training objectives.

[1] https://codecarbon.io/

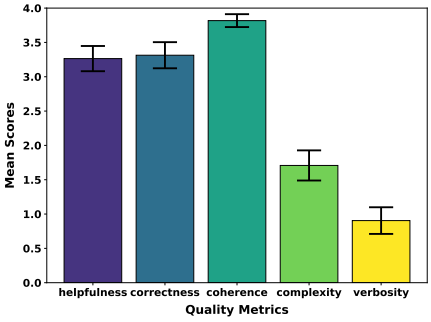


Figure 39: The figure shows the evaluation of preferred responses from DPO pairs using the NVIDIA-Nemotron-4-340B reward model. These high-scoring responses, assessed across multiple quality dimensions, serve as positive examples for fine-tuning models to produce human-preferred outputs.

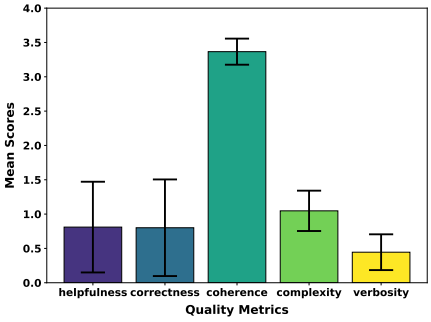


Figure 40: The figure shows the evaluation of rejected responses from DPO pairs using the NVIDIA-Nemotron-4-340B reward model. These lower-scoring responses across multiple metrics serve as negative examples during preference optimization training.

5.4 Graph Retrieval-Augmented Generation (Graph RAG)

Retrieval-Augmented Generation (RAG) utilizes external knowledge databases to assist large language models (LLMs), enabling them to effectively retrieve isolated facts for in-domain question-answering tasks. Graph RAG [11, 17, 18] extends traditional RAG by incorporating structured knowledge graphs for knowledge retrieval. This integration of relational graph

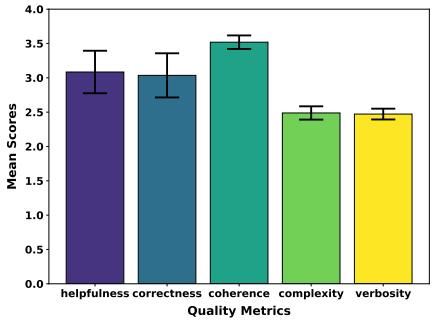


Figure 41: Evaluation of the synthetic SynDIP dataset using the NVIDIA-Nemotron-4-340B reward model. Each chemical process sequence (process context → PFD → PID descriptions) is scored across five key dimensions: helpfulness, correctness, coherence, complexity, and verbosity, validating the alignment with actual process schematics.

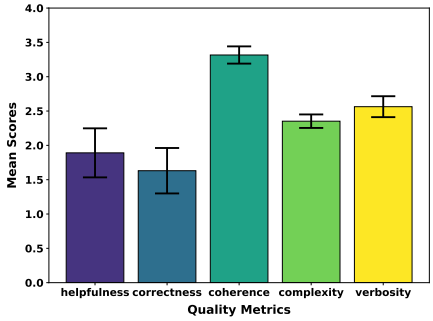


Figure 42: Evaluation of the LogiCore reasoning-augmented dataset using the Nvidia-Nemotron-4-340B reward model. Each multi-step response is scored across five dimensions (helpfulness, correctness, coherence, complexity, and verbosity) to ensure logical validity and faithful representation of PFD/PID schematics.

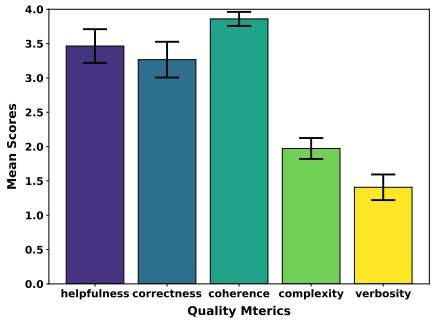


Figure 43: Quality assessment of the Local RAIT synthetic dataset using the NVIDIA-Nemotron-4-340B reward model. Scores across five metrics (helpfulness, correctness, coherence, complexity, and verbosity) demonstrate the quality of retrieval-augmented QA pairs grounded in individual document chunks.

data addresses complex, multi-step reasoning tasks by synthesizing information from multiple, disparate sources. It facilitates multi-hop relationship traversal, enhancing contextual

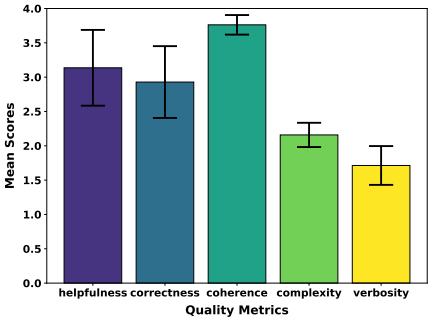


Figure 44: Quality assessment of the Global RAIT synthetic dataset using the NVIDIA-Nemotron-4-340B reward model. The evaluation scores reflect the effectiveness of both detailed and concise answers generated from clustered document chunks, demonstrating robust within-document and cross-document reasoning.

understanding and reasoning processes, which leads to more nuanced, detailed, and accurate responses in open-domain question-answering (ODQA) tasks. As discussed earlier, we employ specialized agents for autonomous web navigation to gather chemical-specific multimodal data from various online sources, focusing on information related to PFDs and PIDs. The aggregated web data is initially stored as documents and subsequently used to populate knowledge graphs. The process begins by constructing property graphs from these unstructured documents, wherein relevant data is extracted and organized into a graph database. To process a document t_i, we first divide its extracted text into smaller chunks using a sliding window technique. Let $C_i = \{c_1, c_2, \ldots, c_M\}$ denote the set of text chunks extracted from t_i, where each chunk c_j is a text segment of size $|c_j|$. We define a window size w and a stride s, applying the sliding window operation such that each chunk c_j spans from position p_j to $p_j + w - 1$, where $p_j = 1 + (j-1) \cdot s$. This method produces overlapping chunks, ensuring contextual continuity. To further enrich these chunks, we use a language model $\mathcal{M}_\theta$ to generate a semantic description $\mathcal{R}_j$ that captures the relationship between chunk c_j and the remaining chunks in C_i. Formally, the relational description is generated as:

$$\mathcal{R}_j = \mathcal{M}_\theta(c_j, C_i \setminus \{c_j\})$$

The generated description $\mathcal{R}_j$ is appended to c_j, forming the enriched chunk $c'_j = c_j \oplus \mathcal{R}_j$. Each augmented chunk c'_j is then represented as a chunk node in the knowledge graph. These enriched chunks, carrying both local and relational context, enhance downstream tasks such as graph-based retrieval and generation. Each augmented chunk c'_j contains structured triples in the form (subject, predicate, object), where subjects and objects are entities linked by relational predicates. The objective is to extract these triples, each representing a distinct semantic relationship. For a given chunk c'_j, the extraction process proceeds as follows: (1) The entities within c'_j are represented as entity nodes, distinct from chunk nodes. Let $E_j = \{e_{j1}, e_{j2}, \ldots, e_{jK_j}\}$ denote the set of entities extracted from c'_j, where e_{jk} represents the k-th entity, and $K_j = |E_j|$ is

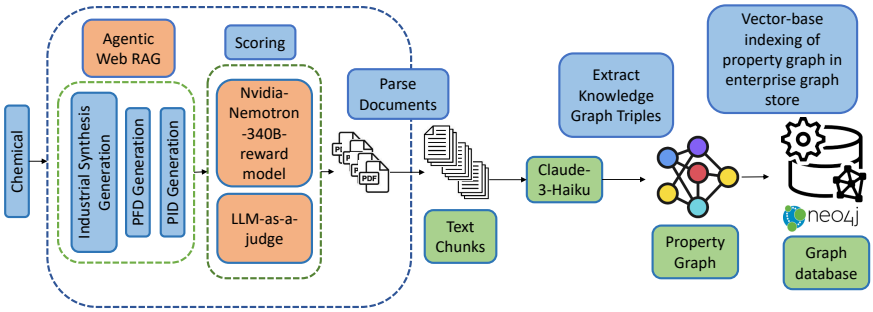


Figure 45: The figure illustrates the end-to-end Graph Retrieval-Augmented Generation (Graph RAG) pipeline for PFD/PID interpretation in chemical process engineering. A multimodal agentic framework—comprising expert agents coordinated by a meta-agent—retrieves and processes data for knowledge graph construction. Unstructured documents are parsed into text chunks, from which knowledge graph triples are extracted and structured into a property graph. The resulting graph is vector-indexed for similarity-based retrieval. Validation leverages LLM-as-a-Judge (GPT-4o) and reward models (NVIDIA Nemotron-4-340B) to optimize knowledge extraction, ensuring factual accuracy, coherence, and task relevance.

the number of entities in the chunk. (2) The relations between entities are represented as directed edges. We define the set of relations (predicates) in chunk c'_j as $\mathcal{R}_j = \{r_{jkm} \mid 1 \leq k \neq m \leq K_j\}$, where r_{jkm} denotes the relation between entities e_{jk} and e_{jm}. Thus, the set of extracted triples from chunk c'_j can be written as:

$$\mathcal{T}_j = \{(e_{jk}, r_{jkm}, e_{jm}) \mid 1 \leq k \neq m \leq K_j\}$$

where each triple $(e_{jk}, r_{jkm}, e_{jm})$ represents a directed relation from e_{jk} to e_{jm} via r_{jkm} within the chunk c'_j. The union of extracted triples from all augmented chunks $C'_i = \{c'_1, c'_2, \ldots, c'_M\}$ constitutes the knowledge graph $\mathcal{G}_i$, where entity nodes are connected by predicate edges. Additionally, each entity e_{jk} is linked to its source chunk node c'_j, establishing provenance relationships. These associations are formalized as:

$$(e_{jk}, \texttt{BELONGS\_TO}, c'_j), \quad \forall e_{jk} \in E_j$$

where BELONGS_TO represents the association relation between an entity and its originating chunk. Thus, the final knowledge graph includes both entity-to-entity semantic triples and entity-to-chunk provenance links, capturing both relational structure and source attribution. The final knowledge graph $\mathcal{G}_i$ is formally defined as a directed graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, where the set of nodes $\mathcal{V}_i$ consists of both chunk nodes and entity nodes. Specifically, the set of nodes is given by:

$$\mathcal{V}_i = \{c'_1, c'_2, \ldots, c'_M\} \cup \{e_{jk} \mid j = 1, 2, \ldots, M \quad \text{and} \quad k = 1, 2, \ldots, K_j\}$$

This set includes chunk nodes c'_j and entity nodes e_{jk} extracted from each chunk. The edges $\mathcal{E}_i$ in the graph consist of two types. The first type, relation edges, captures semantic triples between entities and is defined as:

$$\mathcal{E}_i^{\text{rel}} = \{(e_{jk}, r_{jkm}, e_{jm}) \mid j = 1, 2, \ldots, M \quad \text{and} \quad 1 \leq k \neq m \leq K_j\}$$

The second type, containment edges, represents the association between a chunk c'_j and the entities e_{jk} it contains, and is defined as:

$$\mathcal{E}_i^{\text{cont}} = \left\{(c'_j, e_{jk}) \mid j = 1, 2, \ldots, M \quad \text{and} \quad k = 1, 2, \ldots, K_j\right\}$$

The total set of edges $\mathcal{E}_i$ is the union of relation and containment edges:

$$\mathcal{E}_i = \mathcal{E}_i^{\text{rel}} \cup \mathcal{E}_i^{\text{cont}}$$

Thus, $\mathcal{G}_i$ forms a heterogeneous directed graph containing chunk and entity nodes, linked by relation (semantic) and containment (structural) edges, supporting graph-based retrieval, reasoning, and generation. We improve knowledge retrieval accuracy by identifying and merging duplicate entities that represent the same concept through a two-step approach. Each entity e_{jk} is encoded as a vector embedding v_{jk}, generated using a text-embedding model to capture its semantic representation. To determine whether two entities e_{jk} from chunk c'_j and $e_{j'k'}$ from chunk $c'_{j'}$ refer to the same concept, we first compute the cosine similarity between their vector embeddings:

$$\texttt{sim}(v_{jk}, v_{j'k'}) = \frac{v_{jk} \cdot v_{j'k'}}{\|v_{jk}\| \, \|v_{j'k'}\|}$$

If the similarity score exceeds a predefined threshold τ_{sim}, we further compute the string-based similarity using the normalized Levenshtein distance, which quantifies the minimum number of character edits needed to transform one string into another:

$$\texttt{str\_sim}(e_{jk}, e_{j'k'}) = 1 - \frac{d_{\text{lev}}(e_{jk}, e_{j'k'})}{\max\left(|e_{jk}|, |e_{j'k'}|\right)}$$

where $d_{\text{lev}}(e_{jk}, e_{j'k'})$ denotes the Levenshtein distance between the two strings, and $|e_{jk}|$ and $|e_{j'k'}|$ represent their respective string lengths. Entities e_{jk} from chunk c'_j and $e_{j'k'}$ from chunk $c'_{j'}$ are considered duplicates and merged if both their semantic similarity and string similarity exceed their respective thresholds τ_{sim} and τ_{str}. We apply the hierarchical Leiden algorithm to detect communities C_k at various granularities within the knowledge graph $\mathcal{G}_i$, aiming to optimize modularity M_{Mod}. Modularity measures the quality of

a community structure by comparing the density of intra-community edges to the expected density if edges were placed randomly while preserving node degrees. It is defined as:

$$M_{\text{Mod}} = \frac{1}{2n} \sum_{i,j} \left[\mathcal{A}_{ij} - \frac{d_i d_j}{2n} \right] \delta(c_i, c_j)$$

where $\mathcal{A}_{ij}$ is the adjacency matrix (1 if an edge exists between nodes i and j, 0 otherwise), d_i and d_j are the degrees of nodes i and j, respectively, and n is the total number of edges in the graph. The term $\frac{d_i d_j}{2n}$ represents the expected number of edges between i and j under the configuration model. The function $\delta(c_i, c_j)$ is the Kronecker delta, equal to 1 if nodes i and j belong to the same community and 0 otherwise. A community $C_k = (\mathcal{V}_{C_k}, \mathcal{E}_{C_k})$ is a subgraph where the nodes $\mathcal{V}_{C_k} \subseteq \mathcal{V}_i$ and edges $\mathcal{E}_{C_k} \subseteq \mathcal{E}_i$ are more densely connected internally than to nodes outside the community. These communities organize the graph into densely connected subgraphs, typically representing specific topics or contexts. This structure enhances retrieval by scoping searches within relevant communities and facilitates reasoning by grouping related facts necessary for multi-step inference. The hierarchical Leiden algorithm decomposes $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ into L disjoint communities $\{C_k\}_{k=1}^{L}$ through modularity maximization, where $C_k = (\mathcal{V}_{C_k}, \mathcal{E}_{C_k})$ denotes the k-th community subgraph. This optimization proceeds iteratively through three phases: (1) local node reassignment to neighboring communities to improve modularity; (2) aggregation of communities into super-nodes to construct a reduced graph; and (3) repetition of this procedure on the coarse-grained graph until convergence, yielding a hierarchical community structure. For complex reasoning tasks, relevant information often spans multiple communities, necessitating efficient retrieval by identifying communities aligned with query-specific subgraphs. To achieve this, we rank the top-K communities $\{C_1, C_2, \ldots, C_K\}$ based on their cosine similarity to the user query Q and the summaries of relationship paths within each community. Each community C_k is summarized using a language model $\mathcal{M}_\theta$, which encodes its relational edges $\mathcal{E}_{C_k}^{\text{rel}}$ (subject-predicate-object triples) into a summary s_k. The summarization process is formalized as:

$$s_k = \mathcal{M}_\theta(\mathcal{E}_{C_k}^{\text{rel}}) = \arg\max_S P(S \mid \mathcal{E}_{C_k}^{\text{rel}}),$$

where $P(S \mid \mathcal{E}_{C_k}^{\text{rel}})$ is the likelihood of generating a summary S conditioned on the set of relation edges. The summary s_k retains the structural (containment) and semantic (predicate) relationships of the original subgraph. These summaries are then encoded into vector embeddings $v(s_k)$ using a text-embedding model, enabling efficient similarity computation with the query embedding $v(Q)$:

$$\text{sim}(Q, C_k) = \frac{\langle v(Q), v(s_k) \rangle}{\|v(Q)\| \, \|v(s_k)\|}.$$

The top-K communities with the highest similarity scores are selected and combined into a query-specific subgraph $\mathcal{G}_Q = (\mathcal{V}_Q, \mathcal{E}_Q)$, defined as:

$$\mathcal{V}_Q = \bigcup_{k=1}^{K} \mathcal{V}_{C_k}, \quad \mathcal{E}_Q = \bigcup_{k=1}^{K} \mathcal{E}_{C_k},$$

where $\mathcal{V}_Q$ and $\mathcal{E}_Q$ are the union of nodes and edges, respectively, from the selected top-K communities. This subgraph captures dependencies across disparate facts while retaining critical relationships necessary to answer the user query. Communities C_k are precomputed via the Leiden algorithm, ensuring modularity-optimized clustering. The top-K selection scales sublinearly with graph size, as coarse-grained retrieval via community summaries reduces search space before fine-grained traversal. Finally, a language model $\mathcal{M}_\theta$ generates the answer $\hat{A}$ by conditioning on the query Q and the subgraph $\mathcal{G}_Q$:

$$\hat{A} = \mathcal{M}_\theta(Q, \mathcal{G}_Q) = \arg\max_{\hat{A}} P(\hat{A} \mid Q, \mathcal{G}_Q),$$

where $P(\hat{A} \mid Q, \mathcal{G}_Q)$ denotes the likelihood of generating the answer grounded in the retrieved subgraph. Figure 46 visualizes the Neo4j knowledge graph's nodes (chunks and entities) and edges, supporting Graph RAG's reasoning and retrieval process.

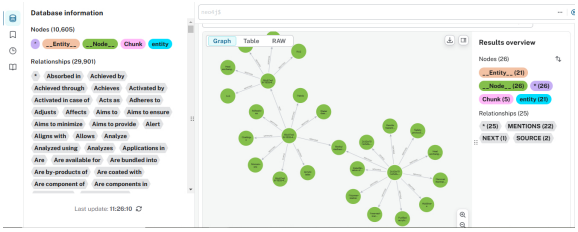


Figure 46: Visualization of the Neo4j knowledge graph constructed for the Graph RAG framework, showing a subset of a larger graph containing 10,605 nodes and 29,901 edges. The graph includes two types of nodes: chunk nodes (text segments enriched with contextual relationships) and entity nodes (named concepts extracted from text). Edges represent `MENTIONS` (linking entities to their originating chunks) and semantic relationships between entities, modeled as subject–predicate–object triples. This structured organization supports multi-hop reasoning and community-based retrieval, enabling the generation of accurate, context-rich descriptions of chemical processes such as PFDs and PIDs.

5.5 Width and Depth Pruning

Transformer-based language models are computationally expensive as Inference Cost $\propto$ Model Size $\times$ (Input Tokens + Output Tokens). Pruning [13, 22, 30, 40, 49–51, 56, 65] removes less critical components, reducing model size and inference costs while preserving accuracy. This enables efficient deployment of smaller, faster, and cheaper models in resource-constrained environments. We consider a small-scale transformer-based language model represented as a parameterized function as follows,

$$\mathcal{F}_\theta : \mathbb{R}^{T \times d_{\text{model}}} \to \mathbb{R}^{T \times V}$$

where T is the sequence length, d_{model} is the hidden dimensionality of the model, and V is the vocabulary size. The model

consists of L stacked transformer blocks $\{\mathcal{T}_\ell\}_{\ell=1}^{L}$, each comprising a Grouped Query Attention (GQA) module, a feedforward network (FFN), and residual connections, all wrapped in pre-layer normalization. GQA separates the number of query and key-value heads. Let H_q and H_{kv} denote the number of query and key-value heads, respectively, with $H_q > H_{kv}$. Let $g = H_q/H_{kv}$ be the number of query heads that share each key-value head, and $d_h = d_{\text{model}}/H_q$ the dimensionality per query head. Given input $\mathbf{X} \in \mathbb{R}^{T\times d_{\text{model}}}$, the linear projections are defined as follows,

$$\mathbf{Q} = \mathbf{X}W^Q \in \mathbb{R}^{T\times H_q\times d_h}, \quad \mathbf{K} = \mathbf{X}W^K \in \mathbb{R}^{T\times H_{kv}\times d_h}, \quad \mathbf{V} = \mathbf{X}W^V \in \mathbb{R}^{T\times H_{kv}\times d_h}$$

For each query head $i \in \{1,\ldots,H_q\}$, its associated key-value head is $k(i) = \lfloor i/g \rfloor$(maps each query head i to its corresponding key-value head by grouping g query heads together per key-value head), and the attention output is as follows,

$$\mathbf{O}_i = \text{softmax}\left(\frac{\mathbf{Q}_i\mathbf{K}_{k(i)}^\top}{\sqrt{d_h}}\right)\mathbf{V}_{k(i)} \in \mathbb{R}^{T\times d_h}$$

The final GQA output is obtained via as follows,

$$\text{GQA}(\mathbf{X}) = \text{Concat}(\mathbf{O}_1,\ldots,\mathbf{O}_{H_q})W^O, \quad W^O \in \mathbb{R}^{d_{\text{model}}\times d_{\text{model}}}$$

Width pruning reduces the intermediate FFN dimensionality d_{ff} by eliminating unimportant neurons. Decoder language models implement FFNs using Gated Linear Units (GLUs), expressed as

$$\text{FFN}(\mathbf{h}) = W_2\left(\phi(W_1^{(a)}\cdot\mathbf{h})\odot(W_1^{(b)}\cdot\mathbf{h})\right)$$

where $W_1^{(a)}, W_1^{(b)} \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}}$, $W_2 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}}$, and $\odot$ denotes elementwise multiplication. Let z_j denote the j-th neuron output from the GLU, and its importance is estimated by

$$I_j = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\left|\frac{\partial\mathcal{L}}{\partial z_j}\cdot z_j\right|\right]$$

where I_j quantifies the average contribution of neuron output (z_j) to the task loss $\mathcal{L}$ over the dataset. Neurons with the lowest I_j values are pruned, reducing the width to $\tilde{d}_{\text{ff}} < d_{\text{ff}}$ by removing rows in $W_1^{(a)}, W_1^{(b)}$ and the corresponding columns in W_2. **Depth pruning** removes entire transformer blocks based on their contribution to the task. For layer $\ell \in \{1,\ldots,L\}$, its importance is computed as

$$I^{(\ell)} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\left|\left\langle\frac{\partial\mathcal{L}}{\partial\mathbf{h}^{(\ell)}},\mathbf{h}^{(\ell)}\right\rangle\right|\right]$$

where $\mathbf{h}^{(\ell)}$ is the residual output of block ℓ. Layers with small $I^{(\ell)}$ values are removed, and the retained set is denoted $\mathcal{S} \subset \{1,\ldots,L\}$ with $|\mathcal{S}| = \tilde{L} \ll L$. For **joint pruning**, we introduce binary gates: $\gamma^{(\ell)} \in \{0,1\}$ for layer retention, and $g_j^{(\ell)} \in \{0,1\}$ for neuron retention in layer ℓ. The forward computation becomes as follows,

$$\mathbf{h}^{(\ell)} = \mathbf{h}^{(\ell-1)} + \gamma^{(\ell)}\cdot\text{FFN}_\ell^{(g)}\left(\text{GQA}_\ell(\text{LN}(\mathbf{h}^{(\ell-1)}))\right)$$

with the gated FFN defined as follows,

$$\text{FFN}_\ell^{(g)}(\mathbf{h}) = \sum_{j=1}^{d_{\text{ff}}} g_j^{(\ell)}\cdot\left[W_2[:,j]\cdot\left(\phi(W_1^{(a)}[j,:]\cdot\mathbf{h})\cdot(W_1^{(b)}[j,:]\cdot\mathbf{h})\right)\right]$$

The constrained optimization objective combines empirical risk and sparsity penalties, where the goal is to minimize task loss while promoting structured sparsity (i.e., layer and neuron pruning) as follows,

$$\min_{\theta,\gamma,g}\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathcal{L}(\mathcal{F}_{\theta,\gamma,g}(\mathbf{x}),y)\right]+\lambda_1\sum_{\ell=1}^{L}(1-\gamma^{(\ell)})+\lambda_2\sum_{\ell=1}^{L}\sum_{j=1}^{d_{\text{ff}}}(1-g_j^{(\ell)})$$

Here, $\mathcal{D}$ denotes the training data distribution, and $(\mathbf{x},y)$ represents an input-output pair sampled from $\mathcal{D}$. The function $\mathcal{F}_{\theta,\gamma,g}$ corresponds to the pruned model, in which specific layers (controlled by $\gamma^{(\ell)}$) and neurons (controlled by $g_j^{(\ell)}$) are selectively removed. The loss function $\mathcal{L}(\cdot,\cdot)$, such as cross-entropy, quantifies the task-specific prediction error. The depth sparsity penalty term, $\lambda_1\sum_{\ell=1}^{L}(1-\gamma^{(\ell)})$, counts the number of pruned transformer layers, where λ_1 is a regularization hyperparameter that governs the penalty for pruning entire layers. Similarly, the width sparsity penalty term, $\lambda_2\sum_{\ell=1}^{L}\sum_{j=1}^{d_{\text{ff}}}(1-g_j^{(\ell)})$, captures the total number of pruned neurons across all FFNs, with λ_2 acting as the corresponding regularization weight that controls the extent of neuron-level pruning. To enable end-to-end differentiability, we relax the binary gates using the Concrete distribution (aka Gumbel-Softmax trick) via a relaxed Bernoulli transformation. Each gate $g_j^{(\ell)} \in [0,1]$ is sampled as:

$$g_j^{(\ell)} = \text{Sigmoid}\left(\frac{1}{\tau}\left(\log\alpha_j^{(\ell)} + \log u - \log(1-u)\right)\right), \quad u \sim \mathcal{U}(0,1)$$

Here, $u \sim \mathcal{U}(0,1)$ denotes a random number uniformly sampled between 0 and 1, and $\sigma(\cdot)$ denotes the Sigmoid function. $\alpha_j^{(\ell)}$ is a learnable logit parameter, and $\tau > 0$ is a temperature hyperparameter that controls the smoothness of the relaxation. A similar sampling strategy is applied to the layer-level gates $\gamma^{(\ell)}$. In short, the joint pruning objective function simultaneously optimizes task performance and model sparsity by minimizing the task loss while penalizing the number of active layers and neurons. Regularization weights λ_1 and λ_2 control the trade-off between accuracy and compression. The optimization learns both the pruned model structure and the corresponding parameters θ. We perform structured pruning during fine-tuning to jointly optimize model performance and sparsity. Pruning is guided by neuron importance scores $I_j = \mathbb{E}\left[\left|\frac{\partial\mathcal{L}}{\partial z_j}\cdot z_j\right|\right]$ and layer importance scores $I^{(\ell)} = \mathbb{E}\left[\left|\left\langle\frac{\partial\mathcal{L}}{\partial\mathbf{h}^{(\ell)}},\mathbf{h}^{(\ell)}\right\rangle\right|\right]$. Differentiable pruning is enabled via continuous gates, where each neuron gate is sampled as: $g_j^{(\ell)} = \text{Sigmoid}\left(\frac{1}{\tau}\left(\log\alpha_j^{(\ell)} + \mathcal{L}_{\text{Gumbel}}\right)\right)$. After training, components with binarized gates $\gamma^{(\ell)}, g_j^{(\ell)} = 0$ are removed, retaining only the most important neurons and layers. This joint optimization achieves model compression through structured sparsity while maintaining downstream task accuracy. In summary, we study structured pruning techniques for transformer-based language models to reduce computational costs while

maintaining accuracy. We propose joint width and depth pruning, where unimportant neurons and layers are removed based on gradient-based importance scores, enabling efficient model compression. Our approach uses differentiable optimization with gating mechanisms, resulting in smaller, faster models suitable for resource-constrained deployment.

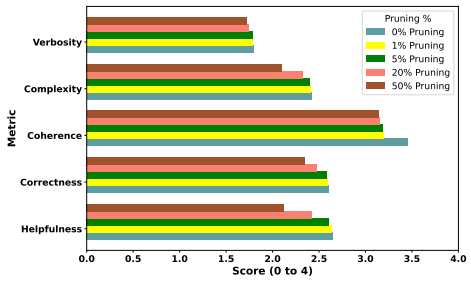
5.5.1 ***The Impact of Width and Depth Pruning on Model Performance: A Qualitative Analysis.*** Figure 47 illustrates the effects of width and depth pruning on model performance on using five qualitative metrics scored from 0 to 4 using pretrained Nemotron-4-340B-Reward model. Both pruning methods generally lead to a decrease in performance scores as the pruning percentage increases. Width pruning (Figure 47a) particularly impacts correctness, complexity, and helpfulness, showing noticeable drops especially at the 20% level, while coherence remains relatively high. Depth pruning (Figure 47b) shows a more pronounced negative effect on coherence and complexity, particularly at higher pruning ratios (20% and 50%). Correctness appears somewhat robust to low levels of depth pruning (1-5%) but declines significantly thereafter. Across both methods, verbosity remains the least affected metric at low to moderate pruning levels. These results highlight the trade-off between model compression and performance, indicating that while pruning can reduce model size/complexity, it comes at the cost of quality, with different pruning strategies impacting specific aspects like coherence or correctness more significantly. Figures 48a(c) and 48b(d) illustrate the impact of width and depth pruning, respectively, on model performance during ChemEval benchmark tasks, which involve PFD/PID generation for unseen chemicals. Both figures demonstrate that increasing the pruning percentage generally degrades the quality of the model's responses across all five metrics. Specifically, width pruning (Figure 48a(c)) shows that higher pruning levels (particularly 20%) lead to significant reductions in correctness and complexity scores. Depth pruning (Figure 48b(d)) also reduces overall quality, with coherence and correctness being notably impacted as pruning percentages increase, showing a steady decline especially at 20% and 50% levels. Across both width and depth pruning applied to these ChemEval tasks, verbosity remained the least affected metric. These results indicate that structural compression via either width or depth pruning hinders the model's ability to effectively generate accurate and coherent PFD/PID descriptions for novel chemical processes. Figures 49e and 49f present the relationship between computational time and the percentage of width and depth pruning applied, respectively. Figure 49e shows that increasing the width pruning percentage correlates with a decrease in computational time: the baseline (0%) took approximately 1350.2 minutes, 5% pruning took 1269.1 minutes, and 20% pruning took 1066.2 minutes. Similarly, Figure 49f indicates that increasing depth pruning percentage also leads to reduced computational time. The baseline time was 1350.2 minutes, decreasing incrementally through 1% (1342.8 min), 5% (1296.2 min), and 20% (1120.7 min), with the most significant reduction observed at 50% depth pruning (796.6 minutes). Both figures illustrate a consistent inverse relationship between the applied pruning percentage and the measured computational time.

5.6 KV Caching and Paged Attention

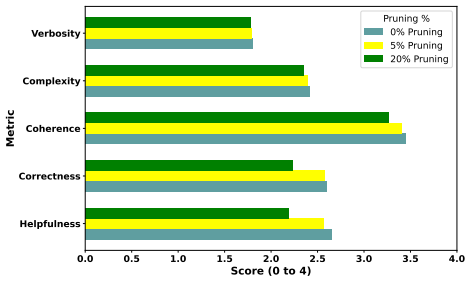
We implement a critical optimization technique for improving the memory efficiency and computational throughput of fine-tuned SLMs during autoregressive decoding (i.e., generating text token-by-token). In autoregressive transformer decoding, at each decoding step i, the model processes the previously available tokens—consisting of (1) the original prompt tokens $\{x_1, \ldots, x_m\}$ and (2) the generated tokens so far $\{x_{m+1}, \ldots, x_{i-1}\}$—and computes a query vector $q_i \in \mathbb{R}^d$. The query attends to all previously processed tokens via their cached key vectors $k_j \in \mathbb{R}^d$ and value vectors $v_j \in \mathbb{R}^d$, where $j = 1, \ldots, i-1$. The attention mechanism computes a weighted sum over the values based on the query-key interactions:

$$\text{Attention}(q_i, K, V) = \sum_{j=1}^{i-1} \text{softmax}\left(\frac{q_i^\top k_j}{\sqrt{d}}\right) v_j$$

Here, $K = [k_1, \ldots, k_{i-1}] \in \mathbb{R}^{(i-1)\times d}$ and $V = [v_1, \ldots, v_{i-1}] \in \mathbb{R}^{(i-1)\times d}$ represent the cached key-value (KV) matrices corresponding to all previously processed tokens. The memory contiguity problem arises because the logical KV cache grows dynamically during decoding, requiring storage of $(i-1) \times d$-dimensional matrices per layer and head at each step i. The linearly growing KV cache in standard autoregressive attention consumes substantial memory, leading to fragmentation and limiting achievable batch sizes. This, combined with its quadratic computational cost—which increases per-token latency—significantly hinders overall throughput. Conventionally, the KV cache is stored contiguously, necessitating pre-allocation of a fixed-size buffer (for a maximum sequence length $L_{\max}$) for each sequence to avoid costly reallocations. However, because sequences vary in length and grow dynamically, this approach is inefficient. It causes internal fragmentation, where memory within an allocated block is wasted when the actual sequence length L is much smaller than $L_{\max}$. More critically, it leads to external fragmentation: when multiple sequences run concurrently, each is allocated a large contiguous block, and as sequences complete at different times, their blocks are freed, leaving gaps of varying sizes between active blocks. Over time, GPU memory becomes fragmented into a patchwork of allocated and free spaces. Even if the total free memory is sufficient, it may be split into small, non-contiguous chunks, causing a new request for a large contiguous block to fail despite sufficient total memory. Furthermore, reallocating the cache for sequences exceeding the pre-allocated buffer size incurs significant $O(L)$ time and memory overhead. These combined inefficiencies reduce achievable batch sizes and degrade overall throughput during model serving. To address the memory inefficiencies of traditional KV caching, PagedAttention [24, 35, 38] adapts the virtual memory paging paradigm from operating systems. Instead of requiring contiguous GPU memory allocations per sequence—which exacerbates fragmentation—it employs a block-based KV cache management strategy.

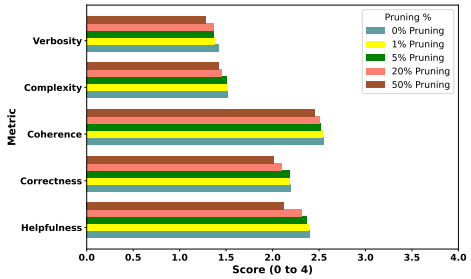


(a) Impact of width pruning percentage on fine-tuned model performance, evaluated using reward model scores (0-4 scale for helpfulness, correctness, coherence, complexity, verbosity) on the 1.5K QA-pair generalization benchmark. Increasing pruning degrades most metrics, particularly correctness and helpfulness.

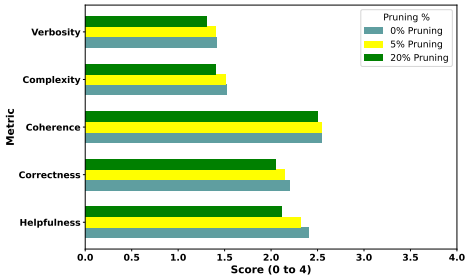


(b) Impact of depth pruning percentage on fine-tuned model performance, evaluated using reward model scores (0-4 scale for helpfulness, correctness, coherence, complexity, verbosity) on the 1.5K QA-pair generalization benchmark. Increasing pruning leads to performance decline.

Figure 47: Evaluation of width (a) and depth (b) pruning impact on fine-tuned model quality. Performance is measured using reward model scores across five dimensions on the 1.5K QA-pair generalization benchmark, showing trade-offs between model compression and response quality.

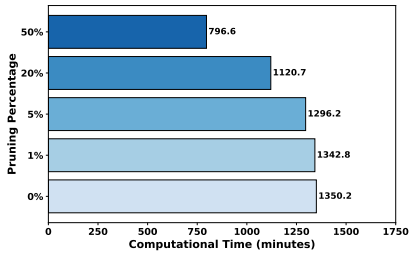


(a) Impact of width pruning percentage on fine-tuned model performance for PFD/PID generation, evaluated using reward model scores (0-4 scale) on the ChemEval benchmark. Increasing pruning degrades quality.

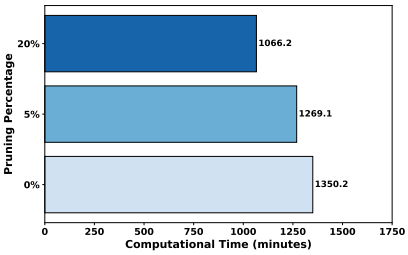


(b) Impact of depth pruning percentage on fine-tuned model performance for PFD/PID generation, evaluated using reward model scores (0-4 scale) on the ChemEval benchmark. Performance declines as layers are removed.

Figure 48: Evaluation of width (c) and depth (d) pruning impact on fine-tuned model quality for zero-shot PFD/PID generation. Performance is measured using reward model scores on the ChemEval benchmark, illustrating the effect of compression on specialized task performance.



(a) Reduction in evaluation computational time (minutes) as a function of width pruning percentage applied to the fine-tuned model.



(b) Reduction in evaluation computational time (minutes) as a function of depth pruning percentage applied to the fine-tuned model.

Figure 49: Impact of width (e) and depth (f) pruning on computational efficiency during evaluation. Plots show the reduction in runtime (minutes) as pruning percentage increases, demonstrating the potential for faster inference with compressed models.

Each sequence's key-value cache is partitioned into fixed-size KV blocks, where each block stores the keys and values for a contiguous segment of B tokens. By decoupling logical blocks from physical memory addresses, PagedAttention allows non-contiguous block allocation, reducing both internal and external fragmentation. This enables dynamic memory reuse across sequences and significantly improves GPU memory utilization, especially for variable-length sequences in batched inference. We can formally define the j-th KV block as:

$$K_j = [k_{(j-1)B+1}, ..., k_{jB}] \in \mathbb{R}^{B\times d}, \quad V_j = [v_{(j-1)B+1}, ..., v_{jB}] \in \mathbb{R}^{B\times d}$$

The innovation lies in the per-sequence block table that maps logical block indices to physical memory locations. This indirection enables three key features: (1) non-contiguous storage where blocks can reside anywhere in GPU memory, (2) on-demand allocation where physical blocks are allocated only when needed, and (3) memory sharing where multiple sequences can reference the same blocks (particularly useful for shared prompts). Rather than computing attention over all past $i-1$ tokens, PagedAttention reformulates this as a block-wise computation. For token position i, the attention output o_i is computed as:

$$o_i = \sum_{j=1}^{\lceil i/B \rceil} \text{softmax}\left(\frac{q_i^\top K_j}{\sqrt{d}}\right) V_j$$

Here, the softmax is computed globally across all blocks up to $\lceil i/B \rceil$, ensuring identical results to standard attention. The score matrix for each block j is $A_{ij} = q_i^\top K_j/\sqrt{d} \in \mathbb{R}^B$, and the softmax normalization is applied over the concatenated scores of all blocks. The implementation achieves this through three optimizations: (i) block-aware memory access patterns, (ii) physical location resolution via block tables, and (iii) prefetching for consecutive blocks. This design provides significant advantages by eliminating both internal fragmentation (through fixed-size blocks) and external fragmentation (via non-contiguous allocation). It enables memory sharing through copy-on-write semantics for shared prefixes while maintaining identical outputs to standard attention. The result is dramatically improved memory utilization that enables larger batch sizes, longer sequence handling, and better overall throughput—critical benefits for production serving systems. While PagedAttention addresses memory fragmentation and enables non-contiguous block-level key-value (KV) caching, it does not reduce the per-parameter memory footprint, as each key and value vector is typically stored in high-precision formats (e.g., FP32 or FP16) that remain memory-intensive. To further improve efficiency, we adopt group-wise quantization for KV cache compression—a technique that reduces memory usage per parameter without requiring retraining during autoregressive decoding. In this approach, each cached KV block—comprising the key matrix $K_j \in \mathbb{R}^{B\times d}$ and value matrix $V_j \in \mathbb{R}^{B\times d}$—is partitioned into column-wise groups, with each group quantized independently using its own scaling factor α_g and zero-point z_g. The quantization process for a group g in K_j is defined as:

$$\tilde{K}_j^{(g)} = \text{round}\left(\frac{K_j^{(g)}}{\alpha_g} - z_g\right), \quad K_j^{(g)} \approx \alpha_g \cdot \left(\tilde{K}_j^{(g)} + z_g\right)$$

where $\tilde{K}_j^{(g)} \in \mathbb{Z}^{B\times d_g}$ represents the quantized integers (typically INT4/8), and the original values are approximated via $K_j^{(g)} \approx \alpha_g \cdot \left(\tilde{K}_j^{(g)} + z_g\right)$. The same procedure applies to V_j. To maintain model fidelity, we incorporate second-order Hessian information during quantization, which captures the curvature of the loss landscape. This allows aggressive quantization of low-curvature weights (tolerating higher error) while preserving high-curvature ones more precisely. The Hessian-aware approach enables accurate low-bit (e.g., 4-bit) quantization with minimal performance degradation. The fixed block structure of PagedAttention ensures efficient dequantization during inference, as each block's group-wise metadata (scales α_g and zero-points z_g) is stored contiguously and accessed predictably. Together, PagedAttention and group-wise quantization provide complementary benefits: PagedAttention eliminates memory fragmentation through non-contiguous block management, while quantization reduces the per-parameter memory footprint by up to 4× (e.g., with 4-bit precision). This combination enables larger batch sizes, longer sequence support, and higher inference throughput—critical advantages for deploying large language models efficiently on memory-constrained hardware.

5.6.1 ***Inference Efficiency Gains with Paged Attention.*** In this section, we evaluate the inference-time efficiency gains enabled by PagedAttention combined with KV cache quantization. By managing the Key-Value (KV) cache in non-contiguous, fixed-size blocks, this approach mitigates the internal and external memory fragmentation inherent in standard contiguous caching and significantly improves inference performance. Since PagedAttention is an inference-only optimization that preserves model output quality, we focus exclusively on system-level metrics, distinct from quality measures such as BLEU, ROUGE, or reward scores discussed in other sections. The efficiency metrics evaluated include inference throughput (tokens generated per second), maximum batch size (largest number of parallel sequences processed), peak GPU memory usage (in GB), and average per-sequence latency (generation time in seconds). We benchmarked our best-performing fine-tuned model, LLaMA-3.2 1B (with fine-tuning and Graph RAG, Variant A), on an NVIDIA H100 GPU using a 500-example subset of the held-out 1.5K QA-pair generalization benchmark dataset. The results, shown in Figure 50, demonstrate significant efficiency improvements. PagedAttention enabled an approximately 2.0× increase in maximum batch size (e.g., 16 versus 8) and improved inference throughput by nearly 1.8× (e.g., 100 vs. 55 tokens/sec) compared to the baseline. While the LLaMA-3.2 1B model itself requires only 2.3 GB of VRAM in FP16 precision, the larger batch size with PagedAttention increased peak GPU memory usage slightly (4.8 GB vs. 4.5 GB) due to greater sequence parallelism. Nonetheless, memory utilization was substantially more efficient because of reduced

fragmentation. The average generation latency for a 2048-token sequence was approximately 39.8 seconds, with only a marginal increase (5–10%) attributable to block management overhead. These findings highlight the practical benefits of PagedAttention for serving fine-tuned SLMs, particularly in RAG-based applications like PFD and PID generation, which involve long and variable-length contexts. This technique complements model-centric strategies such as fine-tuning and pruning, contributing to a more scalable and efficient framework for real-world engineering deployments.

5.7 Low-Latency LLM Decoding Strategies

Let $\mathcal{V} = \{1, 2, \ldots, |\mathcal{V}|\} \subset \mathbb{Z}_{>0}$ denote the vocabulary of a causal language model M parameterized by θ, where $|\mathcal{V}|$ is the vocabulary size. Given a fixed input prompt of length s, $x_0 = (x_1, x_2, \ldots, x_s) \in \mathcal{V}^s$, the objective is to autoregressively generate a target sequence $Y = (y_1, y_2, \ldots, y_T) \in \mathcal{V}^T$ of length T, where each token $y_t \in \mathcal{V}$. The language model defines a conditional probability distribution over the next token:

$$P_M(y_t \mid y_{<t}, x_0; \theta), \quad \text{where } y_{<t} = (y_1, \ldots, y_{t-1})$$

reflecting the causal (left-to-right) nature of the generation process—each token prediction is conditioned only on prior tokens and the fixed prompt. Decoding proceeds greedily (deterministically), selecting the most probable token at each step:

$$y_t = \arg\max_{v \in \mathcal{V}} P_M(v \mid y_{<t}, x_0; \theta)$$

leading to decoding latency that scales linearly with the sequence length T. To enable parallel decoding, the generation task is reformulated as a system of fixed-point equations. For each position $t \in \{1, \ldots, T\}$, define:

$$F_t(y_t, y_{<t}, x_0) = y_t - \arg\max_{v \in \mathcal{V}} P_M(v \mid y_{<t}, x_0; \theta) = 0$$

This system can be solved using Jacobi iteration, which computes speculative updates in parallel at each iteration based on the previous iteration's estimates. Speculation involves parallel guessing of multiple future tokens without waiting for sequential verification. Verification checks whether these speculative guesses match the outputs that greedy decoding would have produced. Let $k \in \mathbb{Z}_{\geq 0}$ denote the iteration index, and let $y_t^{[k]} \in \mathcal{V}$ be the estimate of token y_t at iteration k. The Jacobi update rule is:

$$y_t^{[k]} = \arg\max_{v \in \mathcal{V}} P_M(v \mid y_{<t}^{[k-1]}, x_0; \theta)$$

where $y_{<t}^{[k-1]} = (y_1^{[k-1]}, \ldots, y_{t-1}^{[k-1]})$. While Jacobi iteration enables parallel updates, speculative tokens generated without sequential verification may introduce inconsistencies, discarding otherwise valid generation paths. Consequently, Jacobi decoding alone lacks convergence guarantees and provides limited empirical speedup. To address these limitations, Lookahead Decoding [12, 31, 63] introduces a hybrid approach that combines speculative Jacobi-based multi-token generation with a structured verification mechanism. At decoding step $t \in \mathbb{Z}_{>0}$, it maintains an internal state enabling parallel speculation and exact verification. The internal state consists of:

- The confirmed output prefix, denoted $o = (o_1, \ldots, o_{t-1}) \in \mathcal{V}^{t-1}$, containing previously generated tokens verified to match greedy decoding.
- A two-dimensional token trajectory window $W \in \mathcal{V}^{N \times L}$, where $N \in \mathbb{Z}_{>1}$ is the number of retained Jacobi iterations and $L \in \mathbb{Z}_{>0}$ is the number of speculative lookahead positions predicted in parallel at each decoding step. Here, $L \ll T$ controls the local speculation horizon. Each entry $W_{r,j}$ corresponds to the token predicted at iteration r for lookahead position j.
- Vertical decoding trajectories, where for each column $j \in \{1, \ldots, L\}$, an N-gram candidate $g_j = (W_{1,j}, W_{2,j}, \ldots, W_{N,j}) \in \mathcal{V}^N$ is formed by traversing W vertically across iterations. Each g_j represents a speculative decoding path rooted at the confirmed prefix o.
- The n-gram candidate pool $C \subset \mathcal{V}^N$, defined as: $C = \{g_j : j \in \{1, \ldots, L\}\}$, which stores all vertical decoding trajectories g_j formed at the current step.

During the lookahead branch, the final row $W_{N,1:L}$ is updated by generating speculative token predictions in parallel across all lookahead positions. For each column $j \in \{1, \ldots, L\}$, the next speculative token $W_{N,j}$ is predicted based on a diagonal context extracted from earlier Jacobi iterations. Specifically, if $j \geq N$, the context incorporates the full available diagonal history; otherwise, it truncates to the tokens available from previous columns:

$$W_{N,j} = \arg\max_{v \in \mathcal{V}} P_M\Big(v \;\Big|\; \Big(W_{\min(N-1,j-1),j-1}, \ldots, W_{1,j-\min(N-1,j-1)}\Big), o, x_0; \theta\Big)$$

Here, $W_{N,j}$ denotes the speculative token predicted at lookahead position j during iteration N. The model selects the most probable token $v \in \mathcal{V}$ conditioned on three components: the confirmed prefix $o = (o_1, \ldots, o_{t-1})$, the input prompt x_0, and a causal diagonal context from the speculative window $W \in \mathcal{V}^{N \times L}$. This diagonal context comprises up to $N-1$ previously predicted tokens, selected by moving upward in iteration index and leftward in lookahead position. Formally, the sequence $(W_{r,j'})$ is constructed by decreasing the row index r from $\min(N-1, j-1)$ down to 1, while simultaneously decreasing the column index j' (the moving index for previous lookahead positions used as context) from $j-1$ to $j - \min(N-1, j-1)$. This structure ensures that only causally valid, already predicted tokens are used, maintaining strict autoregressive dependencies. By performing this update in parallel for all $j \in \{1, \ldots, L\}$, the lookahead branch efficiently generates the entire final row $W_{N,1:L}$ while preserving causal correctness. After updating, the system constructs vertical N-grams $g_j = (W_{1,j}, \ldots, W_{N,j})$ for each lookahead position j, and appends them to the candidate pool C. Each g_j represents a speculative decoding path rooted at the confirmed prefix o. The verification branch retrieves up to G candidates from C that satisfy the continuity condition $g_j^1 = o_{t-1}$. For each

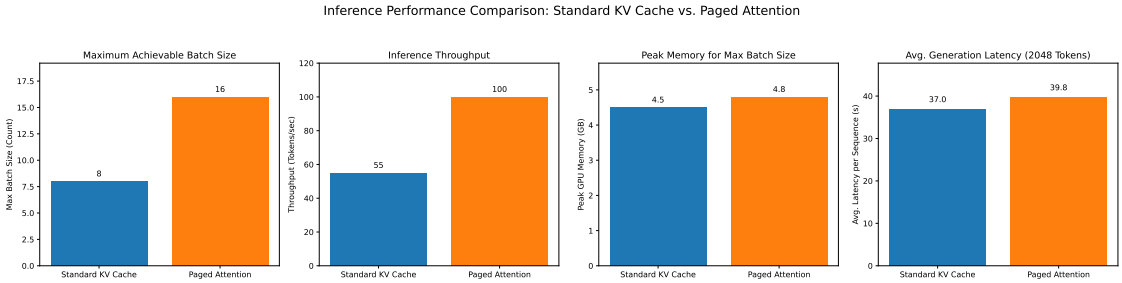


Figure 50: Inference performance comparison between Standard KV Cache and PagedAttention combined with KV cache quantization, presumably on LLaMA-3.2 1B. The figure displays four key metrics: Maximum Achievable Batch Size, Inference Throughput (tokens/sec), Peak GPU Memory (GB) utilized at the maximum batch size, and Average Generation Latency (s) for sequences of 2048 tokens.

candidate $g_j = (g_j^1, \ldots, g_j^N)$, verification proceeds sequentially for $r = 1$ to N:

$$g_j^r \stackrel{?}{=} \arg\max_{v \in \mathcal{V}} P_M\Big(v \;\Big|\; (x_0, o_1, \ldots, o_{t-1}, g_j^1, \ldots, g_j^{r-1}); \theta\Big), \quad \forall r \in \{1, \ldots, N\}$$

If all tokens pass verification, the entire N-gram is appended to o. If a mismatch occurs at some step r, only the verified prefix $(g_j^1, \ldots, g_j^{r-1})$ is retained. The window W is then shifted rightward by the number of accepted tokens, discarding the corresponding speculative entries. This ensures equivalence to standard greedy decoding while enabling speculative parallel generation. Together, the lookahead and verification branches form a hybrid *predict–verify–commit* decoding pipeline. This structure enables speculative multi-token generation while preserving exact output semantics. Lookahead Decoding is a lossless and parallel decoding algorithm that preserves exact output fidelity while significantly reducing sequential decoding steps. It combines token-level Jacobi speculation with n-gram-level greedy verification, organized through a structured two-dimensional window and n-gram cache. This architecture increases FLOPs modestly to reduce decoding latency, scales effectively with parallel compute, and remains fully compatible with the original model without requiring modifications or auxiliary networks.

5.7.1 ***Inference Acceleration with Lookahead Decoding.*** To further reduce generation latency, we evaluated the impact of Lookahead Decoding. This technique accelerates autoregressive generation by predictively generating multiple future tokens in parallel using speculative execution, followed by an efficient verification step against the base model. By enabling multiple tokens to be decoded per single forward pass, it significantly reduces the number of sequential steps required for generation while maintaining the exact output sequence of standard greedy decoding. Our evaluation focused on two key inference speed metrics: generation latency (the total time taken to generate a complete output sequence for one input, measured in seconds) and throughput (the average rate at which tokens are generated, measured in tokens per second). We benchmarked the performance of the LLaMA-3.2 1B fine-tuned model, comparing standard greedy decoding against Lookahead Decoding configured with N=5 iterations and L=10 lookahead positions, as shown in Figure 51. Experiments were conducted on an NVIDIA H100 GPU using the same 500-example subset of the 1.5K QA-pair generalization benchmark from earlier inference tests. The results demonstrate substantial latency improvements. Compared to standard autoregressive decoding, Lookahead Decoding reduced the average per-sequence generation latency for 2048-token sequences from approximately 40.5 seconds to 21.3 seconds, achieving a speedup of nearly 1.9×. Correspondingly, inference throughput increased from approximately 50.6 tokens/sec to 96.1 tokens/sec. While this speedup comes at the cost of increased computational overhead (FLOPs) per decoding step due to parallel speculation and verification, it significantly reduces overall generation time. Unlike speculative decoding methods that require auxiliary draft models, Lookahead Decoding achieves this acceleration using only the original target model. Lookahead Decoding proves particularly effective for SLMs like our fine-tuned Llama-3.2 1B, cutting latency nearly in half while maintaining output quality. The technique's speed gains benefit time-sensitive applications like PFD/PID autogeneration, and combine synergistically with Paged Attention and pruning for greater system efficiency.

5.8 FlashAttention (Optimizing Attention Computation)

FlashAttention [1, 6, 8, 9, 43] improves attention computation by increasing throughput, reducing latency, and lowering memory usage. It is fast, memory-efficient, and exact, producing the same output as standard attention without approximation. We begin by reviewing the standard scaled dot-product attention mechanism. Let $Q \in \mathbb{R}^{N \times d_k}$, $K \in \mathbb{R}^{N \times d_k}$, and $V \in \mathbb{R}^{N \times d_v}$ denote the query, key, and value matrices, respectively, where N is the sequence length, d_k is the key and query dimension, and d_v is the value dimension. The attention scores are computed as $S = \frac{QK^\top}{\sqrt{d_k}}$, where $S \in \mathbb{R}^{N \times N}$ is the pre-softmax logits matrix. An optional mask $M \in \mathbb{R}^{N \times N}$ can be added to S; for causal attention, $M_{ij} = -\infty$ for $j > i$, preventing attention to future positions. Applying the softmax

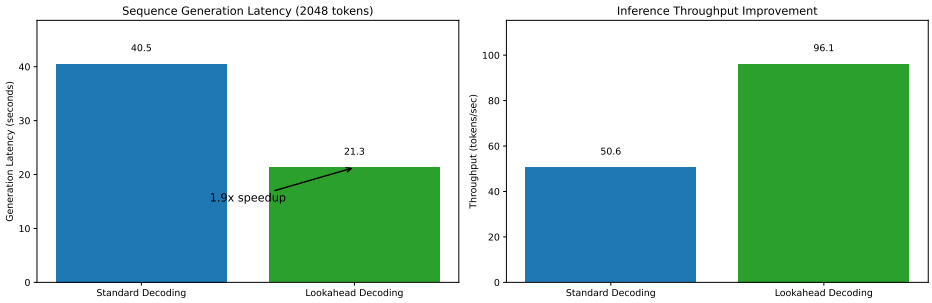


Figure 51: Comparative inference performance metrics (average latency per sequence, throughput in tokens/sec) for the fine-tuned Llama-3.2 1B model using standard greedy decoding versus Lookahead Decoding, highlighting the latency reduction achieved.

operator row-wise produces the attention probability matrix P, where

$$P_{ij} = \frac{\exp(S_{ij})}{\sum_{k=1}^{N} \exp(S_{ik})}$$

Finally, the attention output is computed as $O = PV$, yielding $O \in \mathbb{R}^{N \times d_v}$. However, this standard computation requires materializing and storing the intermediate matrices S and P, resulting in $O(N^2)$ memory usage—a critical bottleneck for large N. Furthermore, significant data movement occurs between high-bandwidth memory (HBM) and on-chip SRAM/registers: Q, K, and V must first be loaded from HBM into SRAM; the matrix multiplication to compute S is performed in SRAM; S must then be written back to HBM if insufficient on-chip storage is available; S must be reloaded from HBM to compute P in SRAM; P must be written to HBM; and finally, P and V must be loaded again to compute the final output O. Altogether, this results in $O(N^2 d_k)$ memory transfers between HBM and compute units, with memory bandwidth becoming the dominant performance bottleneck as N grows. FlashAttention restructures attention computation to eliminate the need to explicitly store the full S and P matrices by partitioning Q, K, and V into manageable blocks and performing an online softmax computation. The matrices are partitioned along the sequence dimension into blocks: Q is split into T_r blocks $\{Q_1, Q_2, \ldots, Q_{T_r}\}$, where each $Q_i \in \mathbb{R}^{B_r \times d_k}$; and K and V into T_c blocks $\{K_1, \ldots, K_{T_c}\}$ and $\{V_1, \ldots, V_{T_c}\}$, respectively, with each block $K_j \in \mathbb{R}^{B_c \times d_k}$ and $V_j \in \mathbb{R}^{B_c \times d_v}$. The block sizes B_r and B_c are chosen such that a single set of Q_i, K_j, V_j, and the intermediate computations fit entirely within SRAM. The approximate memory constraint is: $B_r d_k + B_c d_k + B_c d_v + B_r B_c \ll M$, where M denotes the available SRAM capacity. The algorithm proceeds as follows: for each query block Q_i, the corresponding block output $O_i \in \mathbb{R}^{B_r \times d_v}$, normalization vector $l_i \in \mathbb{R}^{B_r}$, and maximum vector $m_i \in \mathbb{R}^{B_r}$ are initialized to zero, zero, and $-\infty$, respectively. The outer loop iterates over query blocks Q_i, and for each Q_i, the inner loop processes all key-value blocks (K_j, V_j). Each K_j and V_j are loaded into SRAM from HBM, and the local pre-softmax logits matrix is computed as:

$$S_{ij} = \frac{Q_i K_j^\top}{\sqrt{d_k}}.$$

If causal masking is required, entries $S_{ij}[r, c]$ are set to $-\infty$ whenever the absolute position of the query r is ahead of the key c. For numerical stability, the row-wise maximum $m_{ij} \in \mathbb{R}^{B_r}$ across S_{ij} is computed:

$$m_{ij}[r] = \max_{1 \le c \le B_c} S_{ij}[r, c].$$

The shifted exponentials are then calculated:

$$P_{ij}^{\text{hat}} = \exp(S_{ij} - m_{ij}).$$

along with the corresponding row sums:

$$l_{ij}[r] = \sum_{c=1}^{B_c} P_{ij}^{\text{hat}}[r, c].$$

The running softmax statistics are updated block-by-block as:

$$m_i^{\text{new}}[r] = \max(m_i[r], m_{ij}[r]).$$

$$l_i^{\text{new}}[r] = \exp(m_i[r] - m_i^{\text{new}}[r]) l_i[r] + \exp(m_{ij}[r] - m_i^{\text{new}}[r]) l_{ij}[r].$$

The output block O_i is updated accordingly:

$$O_i^{\text{new}} = \frac{\exp(m_i - m_i^{\text{new}}) l_i O_i + \exp(m_{ij} - m_i^{\text{new}})(P_{ij}^{\text{hat}} V_j)}{l_i^{\text{new}}}.$$

After processing each key-value block, the updated values m_i, l_i, and O_i are stored and used for the next iteration. Once all (K_j, V_j) blocks have been processed for a given query block Q_i, the resulting output O_i is written back from SRAM to HBM, and the computation proceeds to the next query block. During the backward pass, storing the full S and P matrices would incur $O(N^2)$ memory overhead. FlashAttention avoids this by recomputing S_{ij} and $\hat{P}_{ij}$ during backpropagation using the saved statistics m_i and l_i. The backward pass follows the same block structure as the forward pass: for each query block Q_i and key-value block (K_j, V_j), we recompute $S_{ij} = Q_i K_j^\top / \sqrt{d_k}$, reapply causal masking if needed, and recover $\hat{P}_{ij}$. Given the incoming gradient $\mathrm{d}O_i$, we compute gradients with respect to V_j as $\hat{P}_{ij}^\top \mathrm{d}O_i$, while gradients for Q_i and K_j are obtained through backpropagation of the softmax using the recomputed $\hat{P}_{ij}$ and normalization statistics. Although this recomputation

requires additional FLOPs compared to standard attention, it eliminates intermediate matrix storage, significantly reducing HBM traffic and improving overall efficiency. As a result, FlashAttention achieves exact equivalence to standard attention while reducing memory requirements from $O(N^2)$ to $O(Nd_k)$, maintaining $O(N^2 d_k)$ computational complexity, and optimizing HBM-SRAM data movement to minimize bandwidth bottlenecks. This approach enables efficient training and inference on long sequences while delivering substantial practical speedups. In summary, the FlashAttention restructures attention computation into a highly I/O-aware algorithm by minimizing HBM–SRAM data movement through blockwise recomputation and online softmax, achieving exact standard attention with drastically reduced memory overhead.

5.8.1 ***Training Acceleration with FlashAttention.*** To further enhance computational efficiency during inference, we incorporated FlashAttention, an optimized attention algorithm designed to reduce memory reads/writes between GPU High Bandwidth Memory (HBM) and on-chip SRAM. By using techniques like tiling, recomputation, and kernel fusion, FlashAttention computes the exact attention output while significantly reducing memory overhead and increasing speed, particularly for long sequences. As an implementation-level optimization, FlashAttention does not alter the mathematical definition of attention and thus has no impact on the model's output quality or task performance metrics (e.g., BLEU, ROUGE, reward scores). Therefore, our evaluation focuses on system-level performance improvements. We benchmarked the training and inference performance using LLaMA-3.2 1B, comparing the standard PyTorch attention implementation against FlashAttention. Experiments were conducted on an NVIDIA H100 GPU. For training, we measured throughput (training examples per second) and peak GPU memory usage during fine-tuning. For inference, we measured throughput (tokens per second). The results, summarized in Figure 52, show substantial gains. During training, FlashAttention doubled the fine-tuning throughput (2.0×, from 8 examples/sec to 16 examples/sec) and reduced peak GPU memory consumption by 15.6% (from 4.5 GB to 3.8 GB). This memory reduction allows for potentially larger batch sizes or longer sequence training within the same memory budget, contributing to the observed throughput increase. During inference, FlashAttention boosted the generation throughput by 1.3× (from 52 tokens/sec to 68 tokens/sec). While not explicitly plotted, this increased throughput typically corresponds to reduced average generation latency. These efficiency improvements stem from FlashAttention's I/O-aware design, which minimizes the costly data movement between HBM and SRAM. This is particularly beneficial for attention computation, which is often memory-bound rather than compute-bound. The integration of FlashAttention complements other optimizations like Paged Attention and Lookahead Decoding. While Paged Attention optimizes KV cache management and Lookahead Decoding reduces sequential generation steps, FlashAttention directly accelerates the core attention computation within each step. Together, these techniques contribute to a highly efficient framework for both training and deploying SLMs for demanding tasks like PFD/PID generation, enabling faster iteration during development and lower operational costs during deployment.

5.9 Test-Time Inference Scaling via Self-Consistency, Confidence-Weighted Entropy, and Self-Reflection

To address limitations in factual accuracy, reliability, and reasoning robustness in small-scale language models (SLMs), we propose a test-time inference scaling mechanism [3–5, 25, 28, 29, 36, 45, 46, 59, 60, 62, 62] that combines three complementary strategies: (1) self-consistency decoding, (2) confidence-weighted entropy scoring, and (3) a self-reflection-based revision mechanism. Unlike fine-tuning or prompt engineering approaches, this method operates purely at inference time, requiring no model parameter updates, and is particularly well-suited for tasks involving multi-step reasoning, such as auto-generation of PFDs and PIDs. First, it involves multiple candidate generation via Chain-of-Thought (CoT) sampling. Given an input query x, the model generates a set of N diverse reasoning trajectories $\mathcal{Y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\}$, where each candidate sequence $y^{(i)} = (y_1^{(i)}, y_2^{(i)}, \ldots, y_T^{(i)}) \in \mathcal{V}^T$ is produced using stochastic decoding (e.g., nucleus sampling or top-k sampling) under a CoT prompting strategy. Here, $\mathcal{V}$ denotes the vocabulary and T is the maximum generation length. At each decoding step t, the token $y_t^{(i)}$ is sampled from the conditional distribution given the input x and the prefix $y_{<t}^{(i)}$ as follows:

$$P_\theta(v \mid x, y_{<t}^{(i)}), \quad \forall v \in \mathcal{V}.$$

where θ denotes the LLM's parameters and $y_{<t}^{(i)}$ refers to the prefix tokens up to decoding step $t-1$. We evaluate the quality of each generated sequence $y^{(i)}$ via a confidence-weighted entropy score, reflecting model uncertainty per decoding step and overall sequence likelihood. At decoding step t, the model provides the predictive distribution $P_t^{(i)}$ over the vocabulary $\mathcal{V}$, conditioned on the input x and prefix $y_{<t}^{(i)}$:

$$P_t^{(i)}(v) = P_\theta(v \mid x, y_{<t}^{(i)}), \quad \forall v \in \mathcal{V}.$$

The entropy of this distribution at decoding step t for candidate i is:

$$H_t^{(i)} = -\sum_{v \in \mathcal{V}} P_t^{(i)}(v) \log P_t^{(i)}(v).$$

where $H_t^{(i)}$ quantifies the model's uncertainty about the token choice at decoding step t. To reflect the relative importance or semantic saliency of each token position in the generated output $y^{(i)}$, we derive weights $\{w_1^{(i)}, \ldots, w_T^{(i)}\}$ using an attention-weighted gradient attribution method. Specifically, let $\alpha_t^{(i)}$ denote the average attention received by the t-th generated token $y_t^{(i)}$ across all attention heads and layers in the LLM. We define the importance weight for decoding step t as:

$$w_t^{(i)} = \frac{\alpha_t^{(i)} \cdot \left| \frac{\partial \mathcal{L}^{(i)}}{\partial e_t^{(i)}} \right|}{\sum_{t'=1}^{T} \alpha_{t'}^{(i)} \cdot \left| \frac{\partial \mathcal{L}^{(i)}}{\partial e_{t'}^{(i)}} \right|} \cdot T.$$

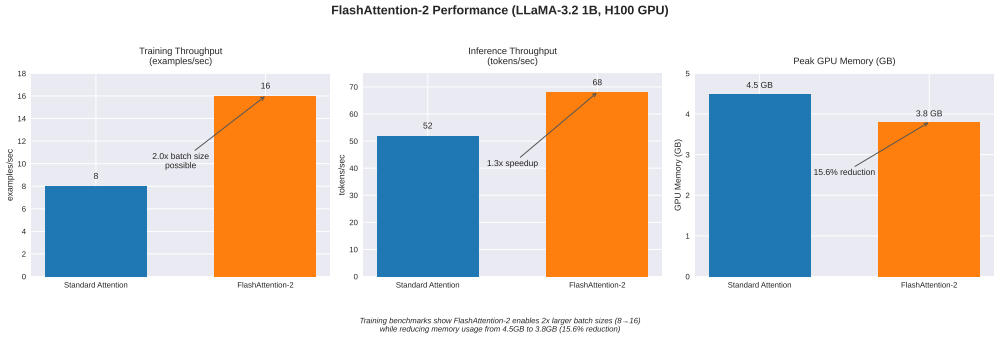


Figure 52: Comparative performance metrics for the Llama-3.2 1B model using standard PyTorch attention versus FlashAttention on an NVIDIA H100 GPU. Plots show training throughput (examples/sec), inference throughput (tokens/sec), and peak training GPU memory usage (GB), highlighting the speed and memory efficiency gains of FlashAttention.

where $\mathcal{L}^{(i)}$ is the negative log-likelihood loss over the candidate sequence $y^{(i)}$, defined as:

$$\mathcal{L}^{(i)} = -\sum_{t=1}^{T} \log P_\theta(y_t^{(i)} \mid x, y_{<t}^{(i)}).$$

Here, $\ell_t^{(i)}$ denotes the model's output logits at decoding step t for candidate i, which are used to compute the gradient $\frac{\partial \mathcal{L}^{(i)}}{\partial \ell_t^{(i)}}$ for the attention-weighted attribution. The gradient term $\left|\frac{\partial \mathcal{L}^{(i)}}{\partial \ell_t^{(i)}}\right|$ captures the sensitivity of the loss to the predicted logits for token $y_t^{(i)}$. This formulation ensures the weights are normalized such that:

$$\sum_{t=1}^{T} w_t^{(i)} = T.$$

These weights $w_t^{(i)}$ provide a profile of token importance over the sequence, influenced by both attention patterns and gradient magnitudes. Using the token-level entropies $H_t^{(i)}$ and the importance weights $w_t^{(i)}$, we compute the weighted average entropy for the full candidate sequence $y^{(i)}$:

$$\bar{H}_w^{(i)} = \frac{1}{T}\sum_{t=1}^{T} w_t^{(i)} \cdot H_t^{(i)}.$$

This metric aggregates token-level uncertainty, giving more significance to uncertainty occurring at decoding steps deemed important by the attention-gradient attribution. Lower values of $\bar{H}_w^{(i)}$ indicate higher confidence, particularly on semantically salient tokens. To measure the overall likelihood of a generated sequence according to the model, we compute the normalized average log-probability:

$$\bar{\ell}^{(i)} = \frac{1}{T}\sum_{t=1}^{T} \log P_\theta(y_t^{(i)} \mid x, y_{<t}^{(i)}).$$

A higher (less negative) value of $\bar{\ell}^{(i)}$ indicates that the sequence is more fluent or probable under the model P_θ. To combine model confidence and fluency, we assign a final score to each candidate $y^{(i)}$ by balancing its weighted entropy and average log-likelihood:

$$\text{Score}(y^{(i)}) = \lambda \cdot \bar{H}_w^{(i)} - (1-\lambda)\cdot \bar{\ell}^{(i)},$$

where $\lambda \in [0,1]$ is a tunable hyperparameter controlling the trade-off between minimizing uncertainty (favoring lower $\bar{H}_w^{(i)}$) and maximizing sequence likelihood (favoring higher $\bar{\ell}^{(i)}$). A lower overall $\text{Score}(y^{(i)})$ indicates a more desirable candidate sequence, reflecting a better balance between confidence and fluency. After computing scores for all N generated sequences in $\mathcal{Y}$, we rank them and select the top-K candidates:

$$\mathcal{Y}_{\text{topK}} = \text{TopK}_{y\in\mathcal{Y}}\left(-\text{Score}(y), K\right),$$

where $\text{TopK}(S,K)$ denotes selecting the K elements with the highest values in set S (corresponding here to the lowest scores after negation). Following the selection of top-K candidates $\mathcal{Y}_{\text{topK}}$, a self-reflection mechanism is applied to enhance reasoning robustness. In the critique phase, an auxiliary model ϕ (which may be identical to θ or a separately fine-tuned model) critiques each candidate $y^{(i)} \in \mathcal{Y}_{\text{topK}}$, generating a critique: $c^{(i)} = \text{Critique}_\phi(y^{(i)})$, aiming to identify logical inconsistencies, missing justifications, or factual inaccuracies in the reasoning trajectory. Subsequently, in the revision phase, the model uses the critique to generate an improved sequence: $y_{\text{rev}}^{(i)} = \text{Reflect}_\phi(y^{(i)}, c^{(i)})$. This yields a set of revised candidates: $\mathcal{Y}_{\text{rev}} = \{y_{\text{rev}}^{(i)} : y^{(i)} \in \mathcal{Y}_{\text{topK}}\}$. For the final aggregation, we form a consensus candidate pool combining the top-K originals and their revisions: $\mathcal{Y}_{\text{cons}} = \mathcal{Y}_{\text{topK}} \cup \mathcal{Y}_{\text{rev}}$. A deterministic extraction function $a(\cdot)$ is then applied to each candidate $y \in \mathcal{Y}_{\text{cons}}$ to retrieve its final proposed answer (e.g., the concluding statement or value), resulting in an answer set: $\mathcal{A} = \{a(y) : y \in \mathcal{Y}_{\text{cons}}\}$. The final output a^* is determined by majority vote over the extracted answers: $a^* = \text{mode}(\mathcal{A})$, selecting the most frequently occurring answer among the candidates. This multi-stage inference process—combining exploration through sampling, confidence-weighted evaluation, targeted self-reflection, and robust consensus selection—significantly improves output reliability without requiring model retraining.

5.9.1 ***Empirical Validation of Test-Time Inference Scaling.*** We empirically validate the efficacy of the proposed test-time inference scaling mechanism for enhancing Small Language Model (SLM) reliability without requiring retraining. Applied to Llama-3.2 1B variants, the mechanism—integrating Chain of Thought sampling (N=4 trajectories), confidence-weighted entropy scoring (λ=0.5), Top-K selection (K=2), model-internal self-reflection, and self-consistency aggregation—was benchmarked against standard greedy decoding on three evaluation datasets: instruction following (DPO), retrieval-augmented QA (RAG), and general QA (SFT). Evaluation was employed using both established NLP metrics and qualitative metrics (Correctness, Coherence, Helpfulness, Complexity, Verbosity), scored via the Nemotron-4-340B reward model. Standard NLP metrics consistently demonstrated performance gains across METEOR, ROUGE, BERTScore, and Similarity metrics on all evaluation datasets (DPO: Figure 53, RAG: Figure 54, SFT: Figure 55), indicating improved lexical and semantic fidelity. Crucially, analysis of the qualitative metrics revealed substantial and consistent improvements in **Correctness** across all tested model configurations (Figure 57), demonstrating the mechanism's effectiveness in reducing factual errors. The fine-tuned models with test-time scaling maintained **Helpfulness** improvements (Figure 58) while preserving baseline **Coherence** levels (Figure 56), indicating stable generation quality. These reliability gains came with expected increases in **Complexity** and **Verbosity** (Figures 59, 60). These results affirm that a synergistic combination of sampling-based exploration, confidence-guided filtering, reflective refinement, and consensus mechanisms at inference time significantly boosts critical quality dimensions, especially factual accuracy, albeit at the cost of increased computational latency and slightly longer, more complex outputs. This trade-off positions the approach as valuable for applications demanding high reliability where the inference cost is permissible.

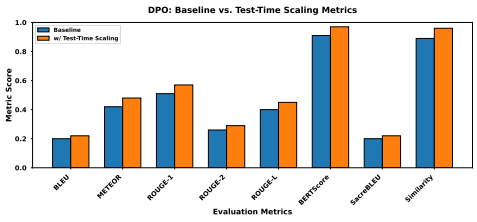


Figure 53: Performance comparison on standard NLP metrics using a fine-tuned Llama-3.2 1B model. The plot contrasts results obtained via baseline greedy decoding (blue) with those obtained using the test-time inference scaling mechanism (orange). Applying the test-time mechanism yields consistent improvements across most metrics (e.g., METEOR, ROUGE variants, BERTScore, and Similarity), indicating enhanced output quality from the same model when using the advanced inference strategy.

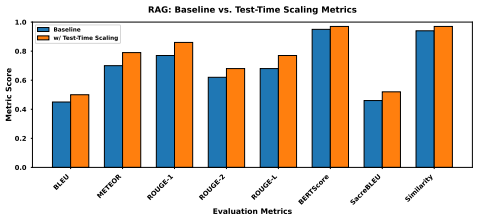


Figure 54: Performance comparison on standard NLP metrics using a fine-tuned Llama-3.2 1B model. The plot contrasts results obtained via baseline greedy decoding (blue) with those obtained using the test-time inference scaling mechanism (orange). The test-time mechanism demonstrates clear benefits, improving scores (particularly ROUGE-1, ROUGE-L, and Similarity) compared to standard decoding on the same model, highlighting its effectiveness.

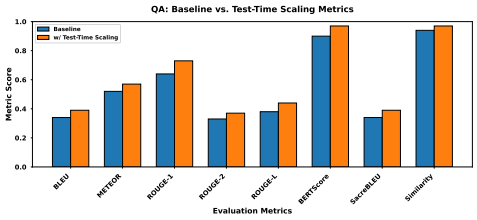


Figure 55: Performance comparison on standard NLP metrics using a fine-tuned Llama-3.2 1B model. Baseline greedy decoding results (blue) are compared against those from the test-time inference scaling mechanism (orange). Applying the test-time mechanism to the same model improves performance across metrics (e.g., ROUGE-1, ROUGE-L, BERTScore, and Similarity), reinforcing the utility of this inference strategy for enhancing SLM robustness post-fine-tuning.

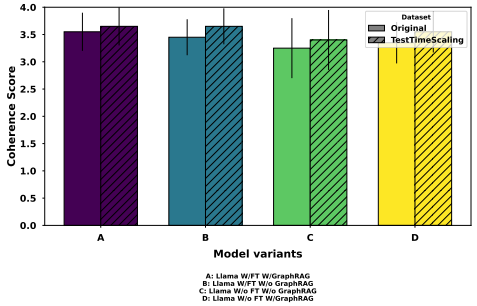


Figure 56: Impact of test-time inference scaling on Coherence Score. Scores are compared between baseline greedy decoding ('Original') and the scaling mechanism ('TestTimeScaling') across four Llama-3.2 1B variants (A: Fine-tuned with GraphRAG, B: Fine-tuned without GraphRAG, C: Base without GraphRAG, D: Base with GraphRAG). The scaling mechanism generally maintains or slightly improves coherence across variants.

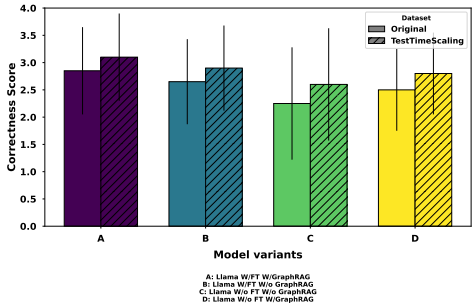


Figure 57: Impact of test-time inference scaling on Correctness Score. Scores are compared between baseline greedy decoding ('Original') and the scaling mechanism ('TestTimeScaling') across four Llama-3.2 1B variants (A: Fine-tuned with GraphRAG, B: Fine-tuned without GraphRAG, C: Base without GraphRAG, D: Base with GraphRAG). The scaling mechanism yields consistent, significant improvements in correctness across all variants.

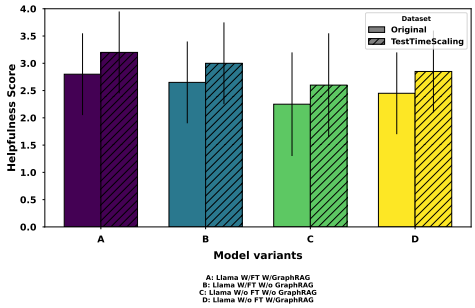


Figure 58: Impact of test-time inference scaling on Helpfulness Score. Scores are compared between baseline greedy decoding ('Original') and the scaling mechanism ('TestTimeScaling') across four Llama-3.2 1B variants (A: Fine-tuned with GraphRAG, B: Fine-tuned without GraphRAG, C: Base without GraphRAG, D: Base with GraphRAG). The scaling mechanism produces notable improvements in helpfulness across all variants.

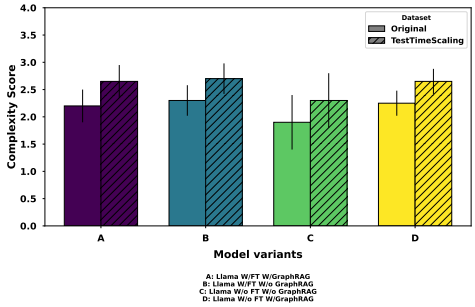


Figure 59: Impact of test-time inference scaling on Complexity Score. Scores are compared between baseline greedy decoding ('Original') and the scaling mechanism ('TestTimeScaling') across four Llama-3.2 1B variants (A: Fine-tuned with GraphRAG, B: Fine-tuned without GraphRAG, C: Base without GraphRAG, D: Base with GraphRAG). The test-time scaling mechanism produces a slight but consistent increase in complexity.

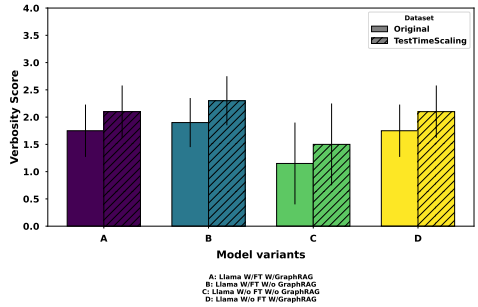


Figure 60: Impact of test-time inference scaling on Verbosity Score. Scores are compared between baseline greedy decoding ('Original') and the scaling mechanism ('TestTimeScaling') across four Llama-3.2 1B variants (A: Fine-tuned with GraphRAG, B: Fine-tuned without GraphRAG, C: Base without GraphRAG, D: Base with GraphRAG). The test-time mechanism results in a slight but consistent increase in verbosity.

5.10 Composite Reward Group Relative Policy Optimization (GRPO)

In our work, we propose a modification to the standard Group Relative Policy Optimization (GRPO) [15, 26, 27, 44] algorithm—originally developed as a reinforcement learning (RL) technique—to directly fine-tune a small-scale language model (SLM). Here, the SLM is treated as a policy network parameterized by θ, where $\theta \in \Theta$ and $\Theta \subset \mathbb{R}^d$ denotes the parameter space. The SLM defines a stochastic, autoregressive policy $\pi_\theta(y \mid x)$, which maps an input prompt $x \in \mathcal{X}$ (from the input space $\mathcal{X}$) to a generated output $y \in \mathcal{Y}$ (from the output space $\mathcal{Y}$). Our goal is to update θ to improve the quality of generated responses relative to a ground-truth reference answer r_x. For each input prompt x with reference r_x, rather than generating a single output, we sample a group of G responses, i.e., $O(x) = \{o_1, o_2, \ldots, o_G\}$, where each o_i is independently drawn from $\pi_{\theta_{\text{old}}}(\cdot \mid x)$, the old policy. This group-level sampling enables assessing outputs on a relative basis. We then assign a composite reward to each generated output o using a weighted sum that integrates three quality metrics. Specifically, the composite reward is defined as

$$r(o, r_x) = 0.3 \cdot r^{\text{rouge}}(o, r_x) + 0.2 \cdot r^{\text{length}}(o, r_x) + 0.5 \cdot r^{\text{LLM}}(o, r_x)$$

where $r^{\text{rouge}}(o, r_x)$ is computed as the ROUGE-L F1 score between the generated response o and the reference answer r_x, capturing semantic and lexical overlap. The term $r^{\text{length}}(o, r_x)$ is defined as

$$r^{\text{length}}(o, r_x) = \begin{cases} \dfrac{\min\big(\text{len}(o), \text{len}(r_x)\big)}{\max\big(\text{len}(o), \text{len}(r_x)\big)} \times 0.5, & \text{if } \text{len}(r_x) > 0, \\ 0, & \text{otherwise} \end{cases}$$

where $\text{len}(\cdot)$ denotes the number of words (or tokens) in a response. $r^{\text{length}}(o, r_x)$ penalizes outputs that are much shorter or longer than the reference, encouraging appropriately sized and complete responses; its value lies between 0 (poor length match) and 0.5 (perfect length match). Finally, $r^{\text{LLM}}(o, r_x)$ is a

numerical score (between 0 and 1) produced by an auxiliary LLM fine-tuned to judge the correctness of o relative to r_x. Consequently, for each generated output o_i in the group $O(x)$, we compute the composite reward $r_i \triangleq r(o_i, r_x)$. To capture the relative performance of outputs for a given prompt, we perform a group-level normalization of these rewards. More precisely, we compute the sample mean:

$$\mu_x = \frac{1}{G}\sum_{i=1}^{G} r_i$$

and the sample standard deviation:

$$\sigma_x = \sqrt{\frac{1}{G}\sum_{i=1}^{G}(r_i - \mu_x)^2}$$

The normalized (or relative) advantage for each output o_i is then defined as:

$$\widehat{A}_i = \frac{r_i - \mu_x}{\sigma_x}$$

This normalization converts raw rewards into z-scores, highlighting outputs that deviate significantly—either positively or negatively—from the group average, thereby guiding the policy update. During fine-tuning, our SLM (i.e., the policy π_θ) generates output sequences token by token in an autoregressive manner. For each output o_i, represented as a sequence $(o_{i,1}, o_{i,2}, \ldots, o_{i,|o_i|})$, we compute the probability of each token using the current policy. We denote by $\pi_\theta(o_{i,t} \mid x, o_{i,<t})$ the probability assigned by the current policy to token $o_{i,t}$, given the prompt x and the preceding tokens $o_{i,<t}$. To ensure training stability, we use an older version of the model, denoted as $\pi_{\theta_{\text{old}}}$, to sample the group $O(x)$. For each token $o_{i,t}$ in each output o_i, we define the probability ratio:

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid x, o_{i,<t})}$$

Using this probability ratio and the previously computed normalized advantage $\widehat{A}_i$, we define the surrogate objective that drives the policy update. Our modified GRPO surrogate objective is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{x\sim\mathcal{X},\\ O(x)\sim\pi_{\theta_{\text{old}}}}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\Big(r_{i,t}(\theta)\cdot\widehat{A}_i,\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\cdot\widehat{A}_i\Big)\Bigg] - \beta D_{\text{KL}}\big(\pi_\theta(\cdot \mid x)\big\|\pi_{\text{ref}}(\cdot \mid x)\big)$$

where ϵ is a clipping parameter that prevents excessive updates when the probability ratio $r_{i,t}(\theta)$ exceeds the bounds $[1-\epsilon, 1+\epsilon]$. The term $\beta\, D_{\text{KL}}(\pi_\theta(\cdot \mid x)\|\pi_{\text{ref}}(\cdot \mid x))$ is a KL divergence penalty that regularizes the updated policy, ensuring it does not deviate too far from a fixed reference policy π_{ref}. The reference policy π_{ref} is fixed throughout training and typically corresponds to a supervised fine-tuned base model. We do not incorporate an explicit entropy regularization term, as the KL penalty sufficiently controls policy drift and diversity. The overall fine-tuning procedure proceeds iteratively as follows. For each input prompt x, we first sample a group of G outputs independently from $\pi_{\theta_{\text{old}}}$, then compute composite rewards $r(o_i, r_x)$ for each output using our weighted combination of ROUGE, length, and LLM-based metrics. These rewards are normalized across the group by calculating the mean μ_x and standard deviation σ_x, yielding relative advantage scores $\widehat{A}_i$ that highlight outputs performing significantly better or worse than the group average. For each token $o_{i,t}$ in every generated output, we compute probability ratios $r_{i,t}(\theta)$ and construct the clipped surrogate objective $J_{\text{GRPO}}(\theta)$. The policy parameters θ are then updated via gradient ascent to maximize this objective. After every optimization step, we synchronize $\theta_{\text{old}} \leftarrow \theta$ for the next iteration. This approach leverages the composite reward signal to holistically improve response quality across all targeted metrics. Crucially, our modified GRPO algorithm operates without a separate value network by: (1) computing composite rewards for each output, (2) normalizing these to obtain relative advantages within each group, and (3) using these advantages to directly optimize the policy through token-level updates. The SLM π_θ thus benefits from efficient, end-to-end reinforcement learning that simultaneously maintains generation diversity and improves performance, all within a computationally lightweight framework.

5.10.1 ***Comparative Analysis of GRPO and SFT for Customizing Language Models on Chemical Process Engineering Tasks***. Figures 62 and 63 depict the training loss progression during the fine-tuning of a Llama 3.2 1B model using Composite Reward Group Relative Policy Optimization (GRPO) on two distinct synthetic dataset types tailored for chemical process engineering. Figure 63 illustrates the loss trajectory for training on QA-style datasets (comprising Factual QA, SynDIP, and LogiCore), designed to enhance domain knowledge and reasoning for PFD/PID interpretation. Figure 62 displays the corresponding loss curve for training on Retrieval-Augmented Instruction Tuning (RAIT) datasets (Local and Global RAIT), focused on grounding model responses in retrieved contextual information. Both plots demonstrate characteristic convergence patterns: a significant decrease in training loss from initial high values during the early epochs, followed by gradual stabilization at lower levels as training proceeds over approximately 10 epochs (Figure 63) and 13 epochs (Figure 62), respectively. This consistent downward trend across both dataset modalities confirms the efficacy of the GRPO algorithm for optimizing the language model on these specialized instruction-following datasets. Figures 61 and 64 evaluate the performance of different fine-tuning strategies, comparing Supervised Fine-Tuning (SFT) and Composite Reward Group Relative Policy Optimization (GRPO) applied to the Llama 3.2 1B and SmolLM2-135M models across five quality metrics assessed by a reward model. On the 1.5K QA-pair generalization benchmark (Figure 61), the GRPO-trained Llama 3.2 1B exhibits the strongest performance in helpfulness and correctness, whereas the SFT-trained Llama 3.2 1B notably excels in coherence. When evaluated on the ChemEval dataset to assess generalization to unseen chemical processes (Figure 64), the GRPO-trained Llama 3.2 1B consistently outperforms both the SFT-trained Llama 3.2 1B and the SFT-trained

SmolLM2-135M across helpfulness, correctness, coherence, and complexity, while all models maintain comparable verbosity. These findings, particularly on the ChemEval generalization benchmark (Figure 64), suggest that GRPO fine-tuning imparts greater robustness and superior performance for the Llama 3.2 1B model in generating descriptions for novel chemical processes compared to standard SFT.

5.11 Verification with Third-Party Chemical Process Simulators

We utilize DWSIM [32], an open-source chemical process simulator, to model material and energy balances under both steady-state and dynamic conditions in chemical plants. DWSIM serves as a comprehensive platform for process design, optimization, and debottlenecking. Process Flow Diagrams (PFDs) typically represent major equipment (e.g., reactors, separators, heat exchangers) and connecting streams, emphasizing the movement of material and energy through the system—specifying flow rates, temperatures, pressures, and other key operating conditions. Piping and Instrumentation Diagrams (PIDs) provide schematic layouts of measurement points (e.g., pressure transmitter PT-101 for vessel pressure monitoring), signal pathways (e.g., dashed lines indicating electrical signals to controllers), and control components (e.g., control valve FV-102 actuated by pressure controller PIC-101). In our framework, DWSIM is employed to construct and simulate PFDs based on autogenerated textual descriptions of novel chemical production processes. While DWSIM does not offer specialized tools for graphical PID creation using standardized instrumentation symbols, it enables functional verification of control strategies derived from PIDs. Users can reconstruct the control architecture within DWSIM's dynamic simulation environment by configuring control loops and signal interconnections, introducing process disturbances, and evaluating closed-loop behavior for controller stability, setpoint tracking, and disturbance rejection. Our framework generates PFDs and PIDs from natural language specifications, defining unit operation sequences, stream flow directions, and control strategies for novel processes. DWSIM is integrated to translate these diagrams into dynamic or steady-state flowsheets, supporting rigorous evaluation of material and energy balances, thermodynamic consistency (e.g., phase equilibria, enthalpy balances), and unit operation performance. Simulations are configured through unit operation blocks (e.g., heat exchangers, distillation columns), thermodynamic models (e.g., Peng–Robinson, NRTL), and stream specifications (temperature, pressure, composition). The system performs phase equilibrium calculations, reaction kinetics modeling, and equipment energy analyses, yielding vapor–liquid distributions, conversion rates, and energy requirements. This simulation-based verification ensures that AI-generated designs adhere to fundamental chemical engineering principles, transforming conceptual proposals into feasible processes and identifying optimization opportunities through first-principles analysis. When framework-generated outputs include instrumentation and control details—such as transmitters, controllers, and final control elements (e.g., "LIC-101 regulates the liquid level in the reboiler sump")—DWSIM enables manual configuration of corresponding feedback control loops using built-in PID controllers. Within its dynamic simulation environment, users can apply setpoint changes and introduce process disturbances to analyze control performance, including loop stability, transient response, and disturbance rejection. This simulation-based workflow supports functional verification of the control strategy implied by the PID, facilitating automated testing, sensitivity analysis, and identification of potential configuration issues such as stream mismatches, non-converging units, or thermodynamic inconsistencies. Figures 65 and 66 present high-level PFDs for nitric acid and sulfuric acid production, respectively. These diagrams were constructed in DWSIM based on textual outputs generated by our framework and manually assembled using DWSIM's unit operation blocks, thermodynamic models, and stream configuration tools. The nitric acid PFD (Figure 65) illustrates a structured sequence of operations, beginning with feed mixing and catalytic oxidation, followed by gas cooling, intermediate conversion, absorption, and final distillation—represented through interconnected unit operations and material flow paths. Similarly, the sulfuric acid PFD (Figure 66) outlines key stages including sulfur combustion, catalytic oxidation, SO_3 absorption, oleum dilution, and product purification, arranged in a logical progression of process units. Figures 67 and 68 illustrate PIDs for the industrial synthesis of nitric acid via the Ostwald process and sulfuric acid via the Contact Process, respectively. Each diagram details key equipment, instrumentation (temperature, pressure, flow, and level sensors), control elements (valves, PID controllers, cascade and feedforward strategies), and piping materials designed to ensure efficient, safe, and regulation-compliant chemical production. These flowsheets reflect realistic industrial workflows and were configured in DWSIM for simulation-based verification. The resulting simulations enable rigorous evaluation of material and energy balances, phase behavior, and equipment performance. By translating language-model-generated flowsheet descriptions into executable DWSIM simulations, we ensure engineering feasibility, identify configuration issues, and support process optimization in accordance with fundamental chemical engineering principles.

5.12 Related Work

This section reviews recent advances in data-driven PFD and PID generation, highlighting their methodologies, limitations, and gaps in industrial applicability. The Generative Flowsheet Transformer [54] introduces a transformer-based model that autocompletes chemical process flowsheets by treating them as linear text sequences using the SFILES 2.0 notation ("a text-based notation encoding topology and unit operations")—a structured, text-based format for representing process flow diagrams. The model is pre-trained on synthetic data and fine-tuned on real flowsheet data, with both datasets converted

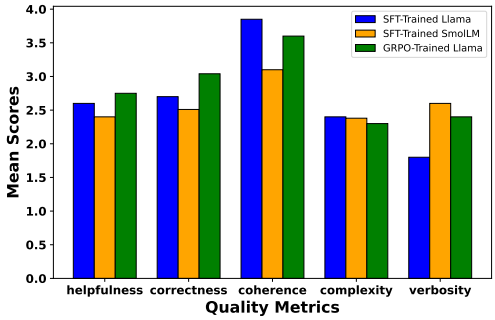


Figure 61: Comparative analysis of GRPO versus SFT fine-tuning on Llama 3.2 1B and SmolLM2-135M models evaluated on a 1.5K QA-pair generalization benchmark. Results show mean performance across five quality dimensions (helpfulness, correctness, coherence, complexity, verbosity), highlighting differences between SFT and GRPO training approaches.

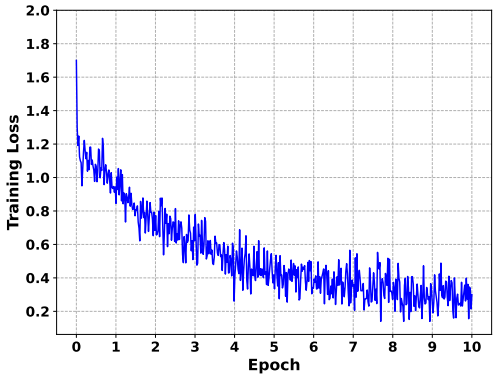


Figure 62: Training loss dynamics for Llama 3.2 1B model fine-tuned with Group Relative Policy Optimization (GRPO) on Local/Global RAIT datasets across training epochs. The convergence behavior demonstrates significant loss reduction over approximately 13 epochs.

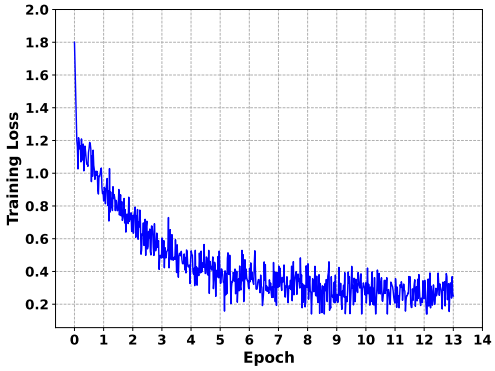


Figure 63: Loss trajectory during GRPO fine-tuning of Llama 3.2 1B on diverse QA datasets (Factual QA, SynDIP, LogiCore). The optimization process exhibits consistent convergence over 10 training epochs.

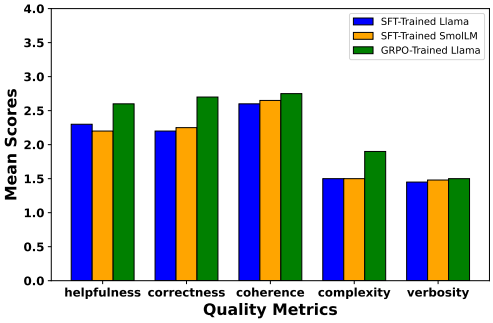
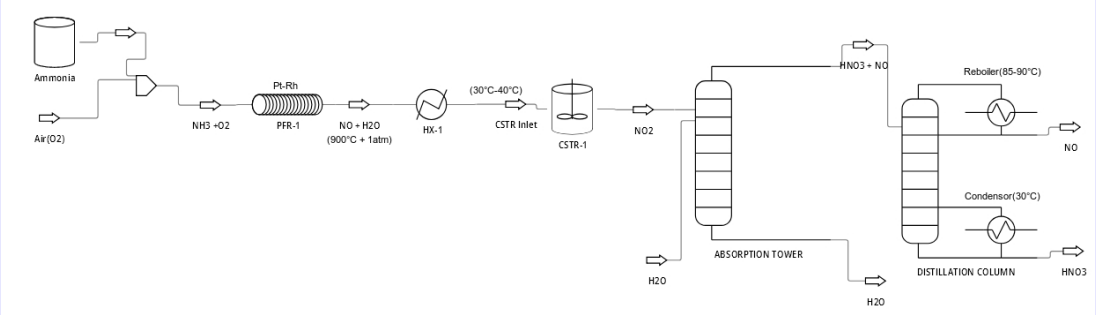


Figure 64: Generalization capability comparison between GRPO and SFT fine-tuning approaches on Llama 3.2 1B and SmolLM2-135M models, evaluated on the out-of-distribution ChemEval dataset. Performance metrics across five quality dimensions reveal GRPO's effectiveness on novel chemical process tasks.

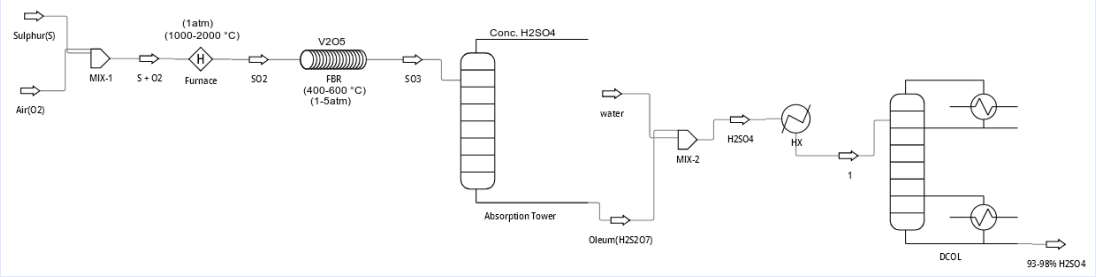
into SFILES 2.0 strings. These strings serve as input for learning the structural grammar of flowsheets, achieving low perplexity while enabling realistic autocompletion. However, the method relies heavily on synthetic data, which poorly reflects industrial variability, and suffers from limited real-world data, leading to unstable generalization. To address the challenge of limited data availability, the Randomized SFILES-based Data Augmentation technique [42] proposes a text-based augmentation method for chemical process flowsheets using SFILES 2.0 notation. This approach introduces an algorithm that applies randomized flowsheet graph traversal and template-based mutations to generate structurally varied (non-canonical) yet semantically equivalent flowsheet strings. The technique supports flowsheet-based process modeling by expanding the diversity of machine-readable training data. However, the method is limited by its dependence on the number of branching points, offering minimal augmentation for small flowsheets while risking overrepresentation of larger flowsheets. Additionally, it only introduces syntactic variations without altering functional or topological features, limiting its ability to improve generalization to structurally novel process designs. The SFILES2Seq framework [19] proposes a data-driven sequence-to-sequence approach for the automatic prediction of control structures, generating PIDs from PFDs. Using the SFILES 2.0 notation, both diagrams are encoded as structured text strings, enabling transformer-based translation. A T5 encoder-decoder model is trained to map PFD sequences to corresponding PID sequences, guided by a custom tokenizer that captures the syntax of unit operations and control elements. The model is first pre-trained on synthetically generated examples, created through a Markov chain-like process that assembles subprocess modules and inserts control structures based on design heuristics. It is then fine-tuned on a small real-world dataset, though performance is limited by dataset size and variability. To improve generalization, augmented SFILES 2.0 strings are generated by varying branching and control unit placements. Beam search is used during inference to produce multiple PID predictions, demonstrating that NLP models can effectively

support automated control structure generation from PFDs. Despite strong performance on synthetic data, the method struggles with real-world generalization due to limited and diverse training samples. The lack of constrained decoding and oversimplified synthetic data further limits its reliability in capturing complex industrial control structures. PID-TALK [2] is a three-stage methodology for enabling natural language interaction with PIDs. First, PIDs are transformed into graph representations that capture both domain hierarchies and lexical interconnections among components. These graphs are then enriched with semantic labels and properties to form labeled property graphs within a knowledge graph framework. Finally, a graph-based retrieval-augmented generation (graph-RAG) approach is employed, where the high-level knowledge graph provides context for large language models, enabling efficient, context-aware querying of PID information while improving interpretability and reducing hallucinations. Despite recent innovations, these approaches exhibit critical limitations that restrict their practical applicability. Current methods are unable to autonomously generate novel industrial PFDs and PIDs, limiting their ability to support new or customized process designs. They often neglect the broader process context—such as operational objectives, feedstock-product relationships, safety constraints, and design rationales—which is essential for producing technically sound schematics. Additionally, many approaches rely heavily on inadequately curated synthetic datasets, failing to capture the complexity and variability of real-world industrial processes. The absence of rigorous simulator-backed validation further compounds these issues, as generated PFDs and PIDs are not tested for operational safety, control robustness, or engineering feasibility, posing significant risks in practical deployment.



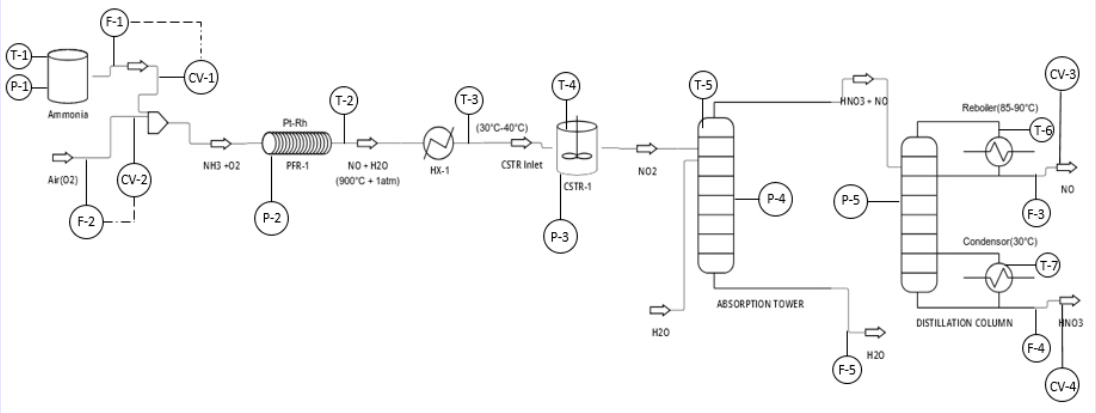
The production of nitric acid (HNO_3) follows a systematic sequence of steps. The process begins with feed preparation, where ammonia (NH_3) from storage and compressed air (O_2) are introduced. Ammonia is stored at ambient temperature and atmospheric pressure, while air is compressed to 1–2 atm. In the next step, ammonia undergoes oxidation in a plug flow reactor (PFR) using a platinum-rhodium (Pt-Rh) catalyst, converting NH_3 and O_2 into nitric oxide (NO) and water vapor at 900°C and 9 atm. The hot gas stream is then cooled to 30–40°C using a heat exchanger (HX1). Nitric oxide (NO) is subsequently oxidized to nitrogen dioxide (NO_2) in a continuous stirred tank reactor (CSTR) at atmospheric pressure and a temperature of 30–40°C. The resulting NO_2 gas is absorbed in water inside an absorption tower, where it reacts to form nitric acid (HNO_3) and nitric oxide (NO) at 60–70°C and 1–2 atm. The nitric acid solution is then purified in a multi-stage distillation column, concentrating it to 60–68% while separating impurities, with the reboiler operating at 85–90°C and the condenser at 30°C. Key operational conditions include maintaining optimal temperatures and pressures in reactors and separation units to enhance efficiency. This optimized nitric acid production process ensures high efficiency, minimizes environmental impact, and is well-suited for large-scale industrial applications.

Figure 65: The figure shows the nitric acid (HNO_3) PFD showing key unit operations (NH_3 oxidation, NO/NO_2 conversion, absorption, distillation) with operating conditions. Generated in DWSIM from framework text.



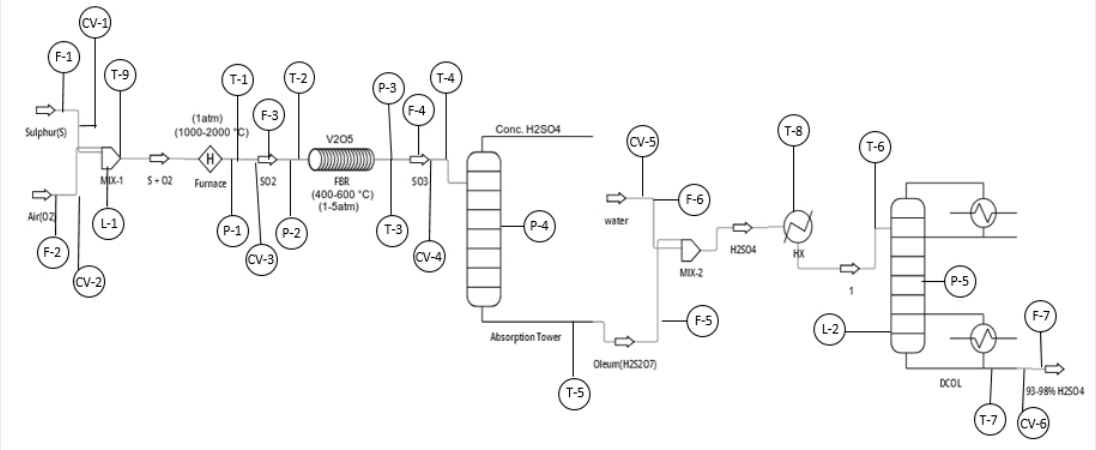
The Contact Process for sulfuric acid (H_2SO_4) synthesis involves several key steps: sulfur combustion, sulfur dioxide oxidation, sulfur trioxide absorption, oleum dilution, and final purification. Initially, elemental sulfur (S) combusts with oxygen (O_2) in a furnace at 1000–1200°C under atmospheric pressure, producing sulfur dioxide (SO_2). The SO_2 then enters a series of fixed-bed reactors, where it undergoes catalytic oxidation with vanadium pentoxide (V_2O_5) at 400–600°C and 1–5 atm to form sulfur trioxide (SO_3). Next, SO_3 is absorbed in concentrated sulfuric acid within a packed absorption tower at 30–60°C, forming oleum ($H_2S_2O_7$). The oleum is then diluted with water in a mixing tank to produce concentrated sulfuric acid. A heat exchanger cools the reactor effluents, and a distillation column purifies the final product, yielding 93–98% pure H_2SO_4. Safety measures include gas detection, automated controls, emergency protocols, and corrosion-resistant materials. Potential bottlenecks include catalyst deactivation in fixed-bed reactors, foaming in absorption towers, and inefficient heat recovery. This optimized process flow ensures efficient, large-scale sulfuric acid production with energy recovery and environmental sustainability.

Figure 66: The figure illustrates the PFD of sulfuric acid (H_2SO_4) production, dynamically simulated in DWSIM. It details critical stages—including sulfur (S) combustion, catalytic SO_2 oxidation, SO_3 absorption, and oleum ($H_2S_2O_7$) dilution—along with associated operating parameters (temperature, pressure, flow rates).



The optimized PID for nitric acid synthesis via the Ostwald process presents a comprehensive layout incorporating essential components, sensors, control mechanisms, and safety systems to facilitate efficient process monitoring and compliance with industry regulations. The system comprises key equipment such as the Ammonia Storage Tank, Air Compressor, Plug Flow Reactor, Heat Exchangers, Continuous Stirred Tank Reactor, Absorption Tower, Distillation Column, Gas Recycling System, and Wastewater Treatment Unit. Instrumentation includes temperature sensors (T-1 to T-7) placed at the ammonia tank, PFR outlet, HX1 outlet, CSTR, absorption tower, and distillation column reboiler and condenser; pressure sensors (P-1 to P-5) at critical points such as the ammonia tank, PFR, CSTR, absorption tower, and distillation column; flow meters (F-1 to F-5) for monitoring ammonia feed, air feed, NO and nitric acid product flow, and wastewater; and level sensors (L-1, L-2) for the ammonia and nitric acid storage tanks. The control infrastructure features valves (CV-1 to CV-4) to regulate ammonia and air feeds, NO, and nitric acid flow, with electric or pneumatic actuators deployed as required. Control strategies employ feedback control via PID controllers to stabilize PFR and CSTR temperatures and pressures, feedforward control to adjust downstream conditions based on upstream flow, and cascade control for distillation column temperature regulation. Recommended piping materials include carbon steel with coatings or stainless steel (e.g., 316L) for ammonia and NO, glass-lined or high-alloy stainless steel (e.g., Hastelloy) for nitric acid, and titanium or stainless steel for heat exchangers handling corrosive streams.

Figure 67: The figure shows the PID for nitric acid production via the Ostwald process, generated using Visual Paradigm Online. The diagram highlights key process units—including the ammonia storage tank, plug flow reactor (PFR), absorption tower, and distillation column—along with instrumentation (temperature, pressure, flow, and level sensors) and control systems (valves, PID controllers, and cascade control). The design reflects process monitoring requirements and compliance with industry standards..



Creating an optimized PID for the synthesis of sulfuric acid via the Contact Process involves integrating best practices, emphasizing critical sensors, control elements, redundancy, reliability, piping materials, and control systems integration. The equipment and piping layout should include a multi-tube furnace for sulfur combustion, a series of fixed-bed reactors with heat exchangers for SO_2 oxidation, and a packed absorption tower with cooling jackets for absorbing SO_3 into concentrated H_2SO_4. A mixing tank for oleum dilution must be equipped with level sensors and flow control for water and oleum, while a heat exchanger is needed for cooling and heat recovery, monitored by temperature and flow sensors. The system should also feature a distillation column with reboiler and condenser controls for sulfuric acid purification, a scrubber system with gas detection for unreacted SO_2, and a filtration system for removing solid impurities. Instrumentation must include temperature sensors (T1 to T9) at critical points such as the furnace outlet, reactor inlets/outlets, absorption tower, distillation column, heat exchanger, and mixing tank. Pressure sensors (P1 to P5) should be installed at the furnace outlet, reactors, absorption tower, and distillation column, while flow sensors (F1 to F7) should monitor sulfur, air, SO_2, SO_3, oleum, water, and final H_2SO_4 flows. Level sensors (L1 and L2) should monitor the mixing tank and distillation column sump. Control valves (CV1 to CV6) must regulate feeds of sulfur, air, SO_2, SO_3, water, and oleum, operated by electric or pneumatic actuators for fast, reliable responses. Control strategies should include feedback control through PID loops for temperature and pressure in critical areas, feedforward control to adjust sulfur and air feed rates based on production goals and data analytics, and cascade control for reactor pressure with temperature as the inner loop. Safety instrumentation is vital. In conclusion, this optimized PID framework for sulfuric acid synthesis via the Contact Process ensures efficient, safe, and reliable industrial-scale production. Incorporating redundancy, advanced control, and real-time monitoring significantly enhances both operational efficiency and safety.

Figure 68: The figure presents the PID for sulfuric acid production via the Contact Process, created using Visual Paradigm Online from framework-generated descriptions. It highlights core equipment including the multi-tube furnace, fixed-bed reactors, absorption tower, and distillation column, along with critical instrumentation (temperature/pressure/flow sensors, control valves) and control strategies (PID loops, feedforward control) for efficient, safe operation.

Dataset type	Prompt
Factual QA dataset	You must generate exactly {n_questions} questions that are strictly and directly related to the specific subtopic provided. No tangential, broad, or off-topic questions are allowed. The subtopic is: {sub_topics} Your response must consist of precisely {n_questions} questions, each directly pertaining to the subtopic, separated by a newline character, with absolutely no additional text, numbering, explanations, or any other characters. Deviation from the subtopic or any failure to generate exactly {n_questions} questions as instructed will result in the output being considered invalid.
DPO dataset	**Chosen Response Prompt Template:** Generate a concise, relevant response to the given question. The response should be directly related to the question, clear, and free of any unnecessary information. It should be helpful, polite, and factually accurate. The question is: {question}. Provide only one response in a plain text, with no additional explanations, introductions, or concluding remarks.
	Rejected Response Prompt Template: Generate a rejected response to the given question that is moderately inaccurate compared to the accurate response. The rejected response may be incomplete or less accurate, but it should still be relevant to the question. The question is: {question} Provide only one response in plain text, with no additional explanations, introductions, or concluding remarks.
LogiCore Dataset	Provide clear, accurate, and concise answers to the following questions. Adhere strictly to the following rules to ensure very high scores in the following categories: (1) **Helpfulness:** Ensure each answer is maximally helpful, fully addressing the question in a way that resolves the query effectively. (2) **Correctness:** Every answer must be factually correct, accurately referencing relevant details from the synthesis description(process context), Process Flow Diagram (PFD), and Piping and Instrumentation Diagram (P&ID). (3) **Coherence:** Ensure that each answer is logically structured and flows smoothly, making it easy for the reader to follow. (4) **Complexity:** Balance complexity appropriately;provide necessary depth without making the answer overly complicated. Ensure the response is insightful where needed. (5) **Verbosity:** Be concise but through. Include all essential details without adding unnecessary information. Ensure that the length of the answer aligns perfectly with the complexity of the question. Failure to adhere to these rules will lead to lower scores and suboptimal performance. Synthesis Description: {synthesis_description} Process Flow Diagram: {pfd_description} Piping and Instrumentation Diagrams: {pid_description} Questions: {questions}
Global/Local RAIT Dataset	Question: {question} Context: {chunk} Provide a concise, accurate, and fact-based answer to the question, using only the information available in the context provided. The answer must be directly derived from the context and should not include any external knowledge, speculation, or interpretation. Ensure that the response is precise and strictly adheres to the content of the context without introducing any additional information.

Table 1: The table shows the Illustrative prompt templates employed within the self-instruct framework to generate distinct synthetic datasets (Factual QA, DPO, LogiCore, RAIT) via teacher LLMs for subsequent instruction tuning.

Dataset type	Prompt
SynDIP Dataset	**Industrial Synthesis generation Prompt Template:** Provide a comprehensive and detailed description of the industrial synthesis process for {chemical_name}. Your description should include: • All key chemical reactions, including reactants, intermediates, and products. • The types of reactors used (e.g., CSTR, PFR) and their operating conditions (e.g., temperature, pressure). • Details of any purification steps, such as distillation, crystallization, or filtration, including the equipment used. • Handling and treatment of by-products and waste streams. • Any recycling loops and the integration of heat exchange systems to optimize energy use. • Specific safety measures taken during the synthesis, especially when dealing with hazardous chemicals. The description should be suitable for an engineer looking to understand the process in detail for implementation in a large-scale industrial setting.
	PFD generation Prompt Template: Based on the following synthesis description, create a detailed textual Process Flow Diagram (PFD) for the synthesis of {chemical_name}. Your PFD should include: • Major equipment involved at each step, such as reactors, heat exchangers, distillation columns, separators, pumps, and compressors. • The flow of raw materials, intermediates, and products through the process, including any recycling streams. • Details of heat integration, such as the use of heat exchangers to recover energy from exothermic reactions or to preheat reactants. • A clear representation of phases (e.g., gas, liquid, solid) in each unit operation, highlighting phase transitions where applicable. • Specific operating conditions at key stages, including temperatures, pressures, and flow rates, to ensure proper operation. • The identification of potential bottlenecks in the process flow, and suggestions for optimizing throughput. Ensure that the PFD is designed according to industry standards and is suitable for scaling up to large-scale production.
	PID generation Prompt Template: Create a detailed Piping and Instrumentation Diagram (P&ID) based on the following process flow diagram (PFD) for the synthesis of {chemical_name}. The P&ID should include: • Detailed placement of sensors (e.g., temperature, pressure, flow, and level sensors) at critical points in the process to ensure precise monitoring. • Specification of control valves, actuators, and control loops required to maintain process parameters within the desired range, including examples of typical control strategies (e.g., feedback, feedforward, cascade control). • Identification of safety instrumentation, such as pressure relief valves, emergency shutdown systems, interlocks, and alarms, to prevent accidents and ensure compliance with safety regulations. • Details of the instrumentation needed for process optimization, such as advanced process control (APC) systems, model predictive control (MPC), and real-time data analytics. • Consideration of redundancy and reliability in the placement of key sensors and control elements to ensure continuous operation and minimize downtime. • Recommendations for the types of piping materials to be used, considering chemical compatibility, temperature, and pressure of the process streams. • Suggestions for the integration of control systems with a distributed control system (DCS) or supervisory control and data acquisition (SCADA) system for centralized monitoring and control. Your P&ID suggestions should align with industry best practices and standards (e.g., ANSI/ISA-5.1) and be suitable for large-scale industrial production.

Table 2: The table shows the Sequential prompt templates used for the multi-stage generation of the SynDIP dataset, covering industrial synthesis descriptions, textual PFD generation, and subsequent textual PID generation.

Dataset Type	Question	Ground Response	Revised Model Response
Factual QA dataset	What are the challenges associated with heat transfer in high-temperature reactors?	Challenges associated with heat transfer in high-temperature reactors include material degradation, reduced thermal conductivity, increased risk of overheating, and the need for advanced cooling systems to manage extreme temperatures effectively.	Key heat transfer challenges in high-temperature reactors include material degradation, potentially reduced thermal conductivity at extreme temperatures, and an increased risk of overheating, often necessitating advanced cooling solutions.
DPO Dataset	How does the scale of production impact reactor design choices?	**Chosen Response:** The scale of production impacts reactor design choices by influencing the type of reactor selected (batch vs. continuous), size and capacity requirements, heat and mass transfer considerations, and the need for automation and control systems to ensure efficiency and consistency at larger scales. **Rejected Response:** The scale of production impacts reactor design choices mainly by determining the reactor's shape and color. Larger scales tend to favor simpler designs, but they do not significantly affect the material selection or technology used.	Production scale heavily influences reactor design, affecting the choice between batch or continuous reactors, determining size and capacity, impacting heat and mass transfer requirements, and dictating the necessary level of automation and control systems.
LogiCore Dataset	What are the implications of reactor type selection (CSTR vs. PFR) on the overall efficiency and yield of the acetone synthesis process?	CSTRs (Continuous Stirred-Tank Reactors) typically offer lower efficiency and yield for acetone synthesis due to backmixing, leading to reactant dilution and reduced conversion. PFRs (Plug Flow Reactors) provide higher efficiency and yield by maintaining concentration gradients, enabling better reactant conversion and selectivity.	Selecting a PFR generally leads to higher efficiency and yield in acetone synthesis because it maintains reactant concentration gradients for better conversion. Conversely, a CSTR often results in lower efficiency due to backmixing and reactant dilution, although it might offer simpler temperature control.
Global RAIT Dataset	How can unreacted hydrocarbons and hydrogen by-products be managed in the synthesis of 1,3-Butadiene?	Unreacted hydrocarbons and hydrogen by-products from the synthesis of 1,3-butadiene can be effectively managed through several strategies. Unreacted hydrocarbons can be recycled back into the reactor or directed to a separate recovery unit for further processing, ensuring that valuable feedstock is not wasted. Hydrogen, on the other hand, can be captured and utilized as a fuel source or for other chemical processes, thereby enhancing overall efficiency and sustainability. This approach minimizes waste and maximizes the utilization of raw materials in the production process.	In 1,3-butadiene synthesis, unreacted hydrocarbons are typically managed by recycling them back to the reactor or separating them for recovery. Hydrogen by-products can be captured and utilized, for instance as fuel or in other chemical processes, which improves overall process efficiency and material utilization.
Local RAIT Dataset	What are the operating conditions for the final synthesis of ammonium sulfate?	The operating conditions for the final synthesis of ammonium sulfate are a temperature maintained at 60-80 degrees Celsius and atmospheric pressure. The reactor type used for this synthesis is a Continuous Stirred Tank Reactor (CSTR).	The final synthesis of ammonium sulfate typically operates at a temperature of 60-80 degrees Celsius and atmospheric pressure, using a Continuous Stirred Tank Reactor (CSTR).

Table 3: Comparison of model responses and ground-truth responses across different synthetic dataset types.