TEACHING LLMS TO UNDERSTAND CODE REPOSITORIES USING SYNTHETIC KNOWLEDGE DATA

Red Hat AI Innovation Team

ABSTRACT

Large language models (LLMs) often struggle to answer questions grounded in complex, domain-specific codebases, especially when deployed in private environments using smaller, open-weight models. These limitations arise from insufficient context understanding, weak reasoning capabilities, and a lack of alignment with specialized terminology or architecture. In this work, we present a synthetic data-driven framework for knowledge infusion using sdg_hub, an open-source tool developed to generate high-quality, document-grounded training data.

Our method transforms curated technical documentation, including annotated code, example notebooks, and code documentation, into synthetic question–answer pairs and reasoning traces using teacher LLMs. We fine-tune Qwen 3 family models on this data.

This targeted fine-tuning approach significantly improves the model's factual accuracy and reasoning performance on repository-specific tasks. Our results demonstrate that small LLMs, when properly customized, can serve as capable domain experts, complementing RAG pipelines while operating in secure, cost-efficient deployments.

1 INTRODUCTION

Understanding and navigating complex codebases is a core challenge for developers working with large language models (LLMs). As repositories grow, with layered abstractions, custom APIs, and domain-specific terminology, the ability to accurately answer code-grounded questions becomes critical. This has fueled interest in **codebase knowledge agents**, which help answer queries by understanding the codebase.

Yet, LLMs—especially those not fine-tuned on code—often struggle in this setting. **Retrieval-augmented generation (RAG)** can surface relevant snippets, but falters when knowledge is fragmented or tied to implicit design patterns. Even accurate retrievals may confuse general programming terms with repository-specific concepts, leading to incorrect answers. These issues are magnified in domains with specialized jargon or unconventional architectures.

The challenge is compounded for teams deploying **small or medium-sized open-weight models** in private environments. These models are typically chosen for their affordability, on-premise deployability, and transparency. However, they lack the reasoning capacity and context length of larger proprietary systems like Claude Code or Cursor. As a result, they often struggle to synthesize information across multiple files or grasp domain-specific logic—leading to hallucinations and brittle answers.

A promising solution is to **customize these smaller models by grounding them in the target codebase**. One emerging approach is to generate high-quality synthetic training data that teaches the model to internalize the repository's structure, design patterns, and domain terminology. We leverage **sdg_hub**, a modular framework for generating document-grounded QA pairs, enabling targeted fine-tuning that teaches the model repository-specific logic and terminology.

In this work, we **customize small open-weight LLMs through knowledge fine-tuning**. While these models may not be strong generalists, we show they can become capable domain experts with focused training.

1.1 Our Contribution

We present a **synthetic data-driven framework for knowledge infusion** using sdg_hub to generate document-grounded QA pairs, reasoning prompts, and instruction data. We fine-tuned models from the **Qwen-3** family on a real-world repository and its documentation.

The resulting model shows improvements in factual accuracy and conceptual understanding, outperforming the base instruct-tuned model. Crucially, our approach is **complementary to RAG**, enhancing response quality even with limited context.

2 Methodology

2.1 Synthetic Reasoning Knowledge Data Generation Using sdg_hub

In this work, we demonstrate how to use sdg_hub for generating synthetic data to customize a large language model (LLM) for answering codebase specific questions. sdg_hub is an open-source synthetic data generation framework developed by the Red Hat AI Innovation team.

sdg_hub allows building sophisticated data generation pipelines with modular design and reusable components. A typical data generation workflow is defined using three components:

- `Blocks`: Modular units for data transformation or teacher prompting.
- `Prompts`: Custom or pre-built templates guiding model behavior within blocks.
- `Flows`: Declarative YAML specifications chaining blocks into multi-stage pipelines.

2.2 Document Curation

We curated two key resources to provide domain-specific context for training:

(1) Documents — primary knowledge sources such as code files, tutorials, and documentation

(2) Document Outlines — brief summaries describing each document's/set of documents content and purpose, giving teacher model extra context.

Our training corpus included:

Codebase files: Each script was thoroughly annotated with detailed docstrings for functions, inline comments for clarity, and high-level summaries describing overall logic and design. Where applicable, we also included architectural notes explaining interactions between modules and classes.

Example notebooks: Notebooks showing how to use existing flows and creating custom flows. All notebooks were annotated and converted to Markdown using an open-source tool.

Documentation and transcripts: We incorporated README files, code documentation, and supplementary materials such as presentation slides and talk transcripts. All transcripts were manually cleaned and verified for clarity and relevance.

2.3 Knowledge Data Generation

We used the `sdg_hub` pipeline to convert raw documents into training data. Instead of using full documents directly, we prompted a teacher model to generate targeted augmentations: key facts, summaries, and extractive overviews, which improved memorization during fine-tuning.

These augmentations were then converted into question–answer pairs, with the teacher model producing both questions and grounded answers. To maintain quality, the pipeline filtered ungrounded QA pairs. We also generated reasoning traces for each QA pair using a reasoning-focused teacher model (Qwen 3 32B), enabling the fine-tuned model to also learn reasoning process while generating the response.

2.4 Data Post-Processing

We do several steps of post-processing for getting the data ready for training. Each training sample combined the (augmented) document with a query, and the model was prompted to generate both a reasoning trace and final answer, in the format:

```
<document>
<query>

<reasoning_trace>
<final_answer>
```

To reduce overfitting on the document we only include maximum 3 QA pairs per document.

This was wrapped in Qwen's chat-style format using role-specific tags (e.g., system, user, assistant) to ensure compatibility with its tokenizer and prompt-handling logic.

2.5 Fine-Tuning Process

We fine-tuned models from the Qwen 3 family, specifically the 32B and 8B variants, chosen for their strong reasoning capabilities and support for tool use. Rather than following traditional instruction tuning, which typically masks out the prompt and backpropagates only on the model's response, we adopted a modified strategy to support deeper knowledge infusion.

In our approach, we masked only the special tokens in the system and chat template (e.g., `<|system|>`), allowing gradients to flow through the document, the query, the reasoning trace, and the final answer. This configuration ensured the model not only learned how to respond but also learned representations of the underlying documents and logic, leading to improved factual accuracy and contextual grounding.

3 Evaluation and Results

To assess the effectiveness of our synthetic data generation and fine-tuning approach, we conducted both **human evaluation** and **benchmark-based evaluation**. Our goal was to measure the model's ability to correctly answer repository-specific questions, both in a closed-book (pure generation) and open-book (RAG-augmented) setting.

3.1 Human Evaluation

To assess the model's understanding of the target code repository, we manually constructed question–answer (QA) pairs designed to probe a range of repository comprehension skills, including code-level details, architectural patterns, and practical usage scenarios.

Each question was paired with its most relevant document segment using a retriever, enabling evaluation in a simulated retrieval-augmented generation (RAG) setup. This allowed us to compare the model's performance in both a closed-book setting (no context provided) and an open-book setting (with retrieved context).

We used GPT-4 as an LLM-based evaluator, comparing model responses against reference answers. The evaluation prompt instructed GPT-4 to assess factual accuracy, faithfulness to the reference, and the presence of hallucinations. Responses were rated on a 1–5 scale, where 5 denoted a fully correct and well-grounded answer, and 1 indicated a completely incorrect or hallucinated response.

As shown in Figure 1, our customized models outperformed the base instruct models across both evaluation modes. Notably, performance gains were consistent across both the 8B and 32B model sizes, with particularly strong improvements in the open-book setting where access to retrieved context reduced hallucinations and improved factual grounding.

These results confirm that our approach leads to substantive improvements in the model's understanding—not just more fluent responses, but more accurate, faithful, and context-aware answers.

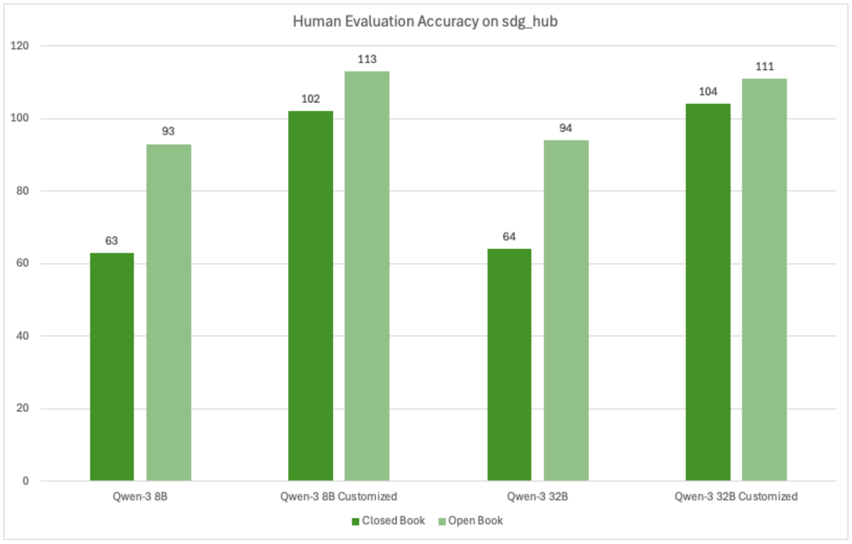

Figure 1: Results of Knowledge Tuning on Codebase QA. We fine-tuned Qwen-8B and Qwen-32B on the sdg_hub codebase and its documentation. The models were evaluated on handcrafted QA pairs using GPT-4 as an LLM judge, with human verification. Evaluation was conducted in both closed-book (query only) and open-book (query + retrieved context) settings.)

3.2 Qualitative Examples

To complement our quantitative results, we examined qualitative examples comparing model responses before and after fine-tuning. Prior to tuning, the base model often produced generic or partially incorrect answers, struggling to capture repository-specific details.

After fine-tuning, responses were notably more accurate, better grounded in the codebase, and structured with clearer reasoning. The model could explain interactions between components and followed domain conventions more reliably. Reasoning traces also improved the logical flow of answers, making them easier to interpret.

These examples highlight how synthetic knowledge and reasoning data can enhance model faithfulness and domain alignment in complex technical settings.