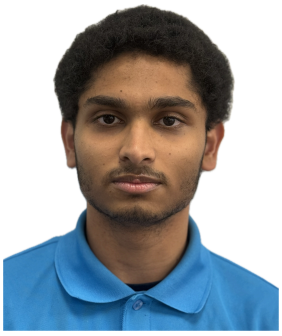
IP 10 – Brown – LLM Compression Trade-off Evaluator

AI 7993 - Section W01 – Fall 2025

Dec 8, 2025

Paul Tuemler
Team Leader



Shivang Patel
Developer / Testing

Alain Kayiranga
Developer / Documentation

Website

https://github.com/skpatel0813/LLM-Compression-Trade-off-Evaluator

Github

https://github.com/skpatel0813/LLM-Compression-Trade-off-Evaluator

Table of Contents

Introduction

Abstract:

This project investigates how model distillation affects the inference quality and power consumption of open-source large language models (LLMs) in code generation tasks. Using Meta's LLaMA as our base model, we evaluate Knowledge Distillation on two benchmark sets, Mostly Basic Python Problems (MBPP) and HumanEval, using pass@k metrics to quantify accuracy. In addition to measuring changes in latency and correctness, we develop an interactive evaluation system and user interface that streamlines the comparison of compression configurations. The resulting framework provides actionable insights into the performance efficiency tradeoffs of distillation and offers practical guidance for deploying resource optimized LLMs.

Overview:

The core of this project is the development of a hybrid inference pipeline that dynamically routes queries between a full-size LLaMA model and its distilled counterpart. By combining distillation with adaptive routing, the system aims to reduce latency and computational overhead while preserving strong performance on code generation tasks. The evaluation centers on code focused benchmarks, specifically MBPP and HumanEval, and uses pass@k as the primary accuracy metric. These results are integrated into an interactive dashboard that visualizes trends in model accuracy, latency, and resource usage, enabling comparison of compression outcomes.

Project Goals:

The main project goals are:

1. Analyze LLM compression techniques, distillation in this case, to measure impact on query accuracy and speed.
2. Develop a dynamic query allocation prototype to intelligently select between full and compressed models
3. Design test cases tailored to code generation and debugging tasks
4. Evaluate performance using pass@k metrics and visualize trends
5. Document best practice for cost efficient LLM deployment

Requirements

Model Compression

Requirements related to generating and managing compression versions of the LLaMA model.

- Support for knowledge distillation to train a compressed “student” model from a full size “teacher” model
- Implement additional compression techniques such as quantization
- Store compressed model in a centralized, version-controlled repository
- Allow users to switch between multiple compressed model versions via configuration settings

Dynamic Query Routing

Requirements for real time routing of incoming queries to appropriate model

- Implementation of routing logic that classifies queries based on complexity
- Route high complexity queries to the full LLaMA model and low complexity queries to the compressed model
- Allow manual override of routing logic through the dashboard
- Ensure query classification adds on more than 50 MS latency overhead
- Maintain logs of all routing decisions for later analysis

Code Specific Test Case Execution

Requirements for generating and running test cases to evaluate model performance.

- Design test datasets for code generation and debugging tasks using open-source repositories, MBPP and HumanEval
- Run test cases on both the full and compressed models
- Calculate pass@k scores for each test case automatically
- Export raw test case results in CSV format

Performance Evaluation

Requirements for measuring and comparing model accuracy and efficiency.

- Track inference latency and response time for both models
- Measure GPU and CPU resource usage during test execution
- Compare full vs. compressed model performance using pass@k and latency metrics
- Generate automated summary reports for each experiment

Reporting Dashboard

Requirements for the visualization and reporting component of the system.

- Display real time graphs of accuracy, latency, and resource consumption
- Provide controls for configuring routing rules and model selection
- Support downloading reports as CSV

Analysis

The evaluation of our hybrid inference framework reveals several important insights into the trade offs between full scale and distilled LLaMA models for code generation oriented tasks. Across both MBPP and HumanEval benchmarks, the 8b LLaMA model consistently delivered the highest overall accuracy and reliability, reaffirming its suitability as a strong baseline for code synthesis. Its performance advantage was most noticeable on more complex prompts, where reasoning depth and precision played a significant role in achieving correct solutions.

The distilled model, however, demonstrated promising efficiency gains while maintaining competitive accuracy. Although it did not surpass the 8B model in raw pass@k metrics, the reduction in inference latency and computational cost made it a compelling option for lightweight workloads. The results suggest that distillation preserves much of the original model's capability while enabling more resource constrained deployment scenarios. The most significant improvements emerged when integrating these models within the hybrid routing system. By dynamically allocating queries between full scale and distilled models, the pipeline pivoted to the strengths of each model. The distilled model handled simpler, low risk tasks quickly, while the 8B model processed more complex queries, anything too complex was passed onto the 70B model although it was overall outperformed by the 8B. In several trials, the combined system achieved higher pass@k scores than the distilled alone, while also decreasing resource consumption when relying exclusively on the full model.

Design

Design Constraints

The design of this system must balance performance, cost, and scalability. Since LLM inference is computationally expensive, compression must significantly reduce resource consumption without compromising accuracy for complex code tasks. Additionally, the dynamic query router must operate with minimal latency overhead to preserve user experience.

Environment

The system is deployed in a cloud-based environment, supporting GPU-accelerated inference. The primary development environment uses Python and frameworks such as PyTorch and Hugging Face Transformers. The dashboard was built with a custom UI and MLFlow for real-time visualization. [Add cloud provider, storage limits, and networking configs]

User Characteristics

The primary users of this system are developers and machine learning engineers focusing on code related problem solving. They are expected to have intermediate to advanced programming knowledge and familiarity with cloud platforms. The system will provide both a dashboard for high level insights and APIs for integration into development workflows.

System

The system comprises three main components:
Model Compression Engine – Implements distillation and other compression techniques to produce smaller models.
Dynamic Query Router – An adaptive logic layer that evaluates query complexity and selects between the full or compressed model.
Reporting Dashboard – Visualizes metrics such as accuracy, inference speed, and resource consumption.

Development

System Architecture

Implements a token-based complexity routing architecture that intelligently distributes queries between compressed and full-scale models

Core components:

- Complexity Estimator
    - Uses tiktoken encoder to estimate query complexity on 1-10 scale based on token count
- Routing Logic
    - Queries less than or equal to 150 tokens are given a complexity score between 1-6, and these queries are routed to the distilled 8B model
    - Queries greater than or equal to 151 tokens are given a complexity score between 7-10, and these queries are routed to the 8B model

Model Preparation

LLaMA model preparation:

- Knowledge Distillation Training
    - Dataset: MBPP (Mostly Basic Python Problems)
        - Contains 974 Python problems with pass/fail test cases
            - Trained with 800 examples
            - Validated with 100 examples
            - Tested on 74 examples
            - Prepared with chat conversation formatting
    - Loaded 70B and 8B models
    - Training Configuration
        - Sequence length: 4096 tokens
        - Epochs: 10
        - Precision: FP16

Routing Mechanism

Rationale: This optimized threshold emerged from empirical testing showing that:

- Distilled model handles most queries that below 150 tokens effectively
- Complex queries require deeper reasoning benefit from 8B model
- Minimizes unnecessary 8B invocations while maintaining performance

Routing Evaluator orchestrates:

- Problem classification by complexity
- Separate evaluations per model-complexity pairing
- Result filtering and combination
- Comprehensive metrics collection via MLflow

Evaluation Pipeline

MBPP Dataset Preparation:

- Downloads from HuggingFace Hub
- Converted to chat conversation format compatible with Llama-3 templates
- Splits: 80/10/10 train/val/test
- Isolated cache directory to avoid conflicts
- Verification checks for data integrity

HumanEval Evaluation

- Enhanced Code Extraction
    - Removes markdown fences (```python```)

    - Strips "assistant" markers from chat format
    - Finds SECOND function definition (actual completion vs prompt echo)
    - Filters docstrings and preamble text
    - Validates Python syntax via ast.parse()
- Generation Pipeline
    - System message emphasizes completing function body only
    - Formatted via tokenizer.apply_chat_template()
    - Max tokens: 512
- GPU Monitoring:
    - Uses pynvml (primary) or nvidia-smi (fallback)
    - Tracks per-GPU metrics every 1 second:
        - Utilization %
        - Memory usage (used/total GB)
        - Power consumption (watts)
        - Temperature (°C)
    - Calculates energy consumption (watt-hours)

Pass@k

- Probability that ≥1 of k generated samples passes all test cases. Provides robustness metric beyond single-attempt accuracy.

User Interface Development

Architecture

Core Components

- app.py (970 lines)
    - Main Streamlit application
    - Model loading with caching (@st.cache_resource)
    - Interactive problem selection and testing
    - Real-time code generation and evaluation
- config.py
    - Model configurations (Student 8B, Distilled 8B+LoRA, Teacher 70B)
    - Path management for models, LoRA adapters, and results
    - Default generation parameters (temperature, tokens, etc.)
- eval_helper.py
    - Test execution using HumanEval framework
    - Handles code extraction and validation
    - Manages temporary files for safe execution

Key Features

Four-Tab Interface

1. Interactive Testing: Single problem generation and testing
2. Model Comparison: Side-by-side model outputs (placeholder)
3. Past Results: View historical evaluation metrics
4. About: Documentation and resource information

Functionality

- Model Selection: Dropdown for Student/Distilled/Teacher models
- Problem Browser: Select from 164 HumanEval problems
- Complexity Routing: Automatic complexity estimation (token-based)
    - ≤150 tokens → Route to Distilled (8B)
    - 150 tokens → Route to Teacher (70B)
- Generation Controls: Adjustable temperature, max tokens, sample count
- Real-time Testing: Execute against HumanEval test cases
- Pass@k Metrics: Calculate pass@1, pass@5, pass@10
- MLflow Logging: Optional experiment tracking

Tooling and Infrastructure

Core Dependencies

- PyTorch ≥2.3: Deep learning framework
- Transformers ≥4.43: HuggingFace model loading
- Accelerate ≥0.33: Multi-GPU distributed training
- Datasets ≥2.20: HuggingFace dataset utilities

Evaluation Tools

- human-eval: Official HumanEval benchmark
- tiktoken: OpenAI tokenizer for complexity estimation

Key Design Decisions

MBPP and HumanEval datasets

- All Python dataset
    - Sponsers were going to use it for Python-based queries
- Contains pass/fail test cases
    - Makes it easier to evaluate all models

Token-Based Routing (Not Semantic):

- Simplicity: No additional ML model for classification

- Speed: O(1) complexity estimation vs transformer forward pass
- Reliability: Deterministic, no inference-time failures
- Transparency: Easy to debug and explain routing decisions

Knowledge Distillation Over Pruning/Quantization Alone:
- Quality: KD transfers "reasoning patterns" from teacher
- Flexibility: Can combine with quantization
- Data Efficiency: Works with limited labeled data (teacher provides soft targets)

HumanEval as Primary Benchmark:
- Standardization: Industry-standard code generation benchmark
- Reproducibility: Official evaluation scripts ensure consistency
- Pass@k Metrics: More robust than single-attempt accuracy
- Compatibility: No changes needed to evaluation pipeline after training

Implementation Challenges

CodeBLEU and BLEU were initially used as metrics to evaluate all three models
- Vague and only showed the similarities in semantics and syntax, not accuracy of the models
    - Switched to Pass@k metrics

CodeSearchNet and OpenCodeInstruct datasets were used initially
- Contained only docstrings, functions, prompts, etc
    - Resulted in Pass@k metrics to be 0
        - Switched to MBPP and HumanEval datasets

Testing and Validation

Agreement Metrics During Training:
- Token Accuracy: Student's top-1 matches teacher's top-1
- Top-5 Agreement: Student's prediction in teacher's top-5
- KL Divergence: Distribution similarity
- Logged every evaluation step to detect degradation

Comprehensive Evaluation Suite:
- Unit-Level: Per-problem pass/fail on HumanEval
- Aggregate: Pass@k metrics across full benchmark
- Cross-Model: Comparative analysis (8B vs 70B vs Distilled Model)
- Energy: GPU power consumption and efficiency metrics

Version Control

All changes to code will be proposed through GitHub branch control and approved by a fellow team member to ensure that no code is being added to the main project without confirming functionality.

As a GPU will be required for the development and testing of our chosen LLM a resource will be necessary. After discussions with the professor, our team settled on using an environment provided by Kennesaw State University.

Summary

This project explored how model distillation and adaptive routing can improve the efficiency of large language models for code generation tasks. Using Meta's LLaMA as the baseline, we developed a hybrid inference system that dynamically routes queries between the 70B, 8B, and distilled model variants. Benchmarking on MBPP and HumanEval with pass@k metrics demonstrated that the 8B model remained the strongest individual performer in terms of accuracy and performance. However, the distilled model offered meaningful reductions in latency and resource usage while retaining a slightly competitive performance. While our research did not surpass the performance of 8B, distillation shows strong promise for further development.

The hybrid routing framework successfully combined the strengths of all models. Simpler tasks were handled efficiently by the distilled model, while more complex queries were delegated to the 8B and 70B models, resulting in improved throughput and stable accuracy. This approach achieved better overall performance than using the distilled model alone and delivered improved efficiency compared to relying solely on a single full sized model.

Overall, the project shows that distillation, when paired with adaptive routing, has the potential to provide a practical and effective way to deploy resource optimized LLMs without significantly compromising code generation quality.

Appendix

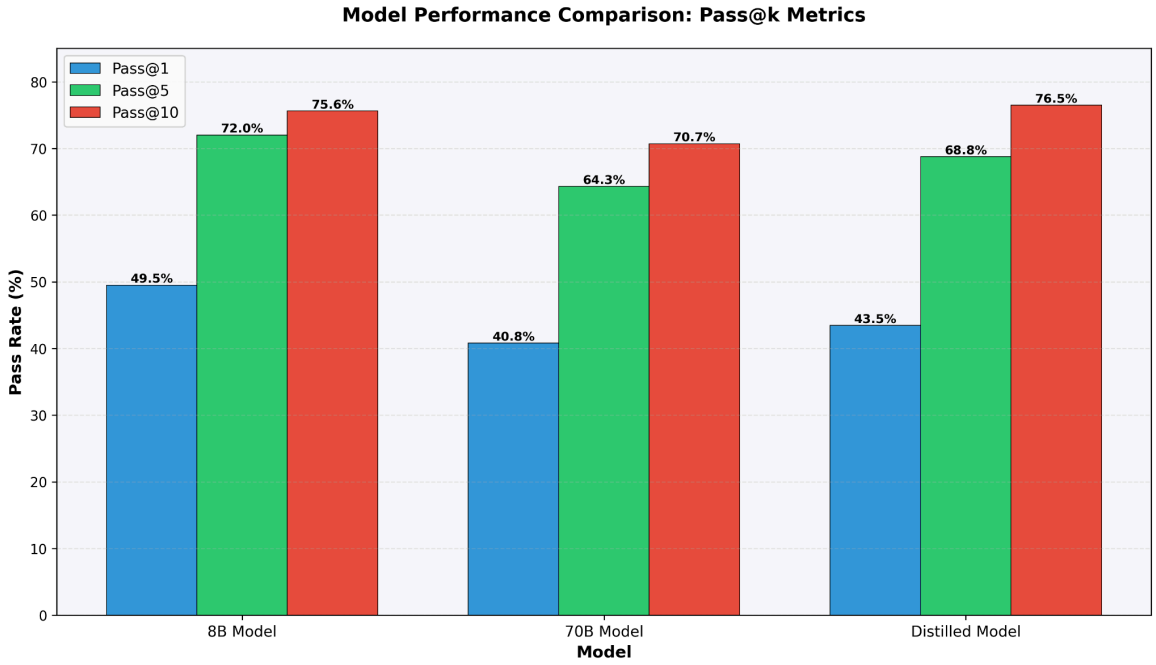


Fig 1. The above graph shows all 3 model performance with pass@k metrics at 1, 5, and 10 passes

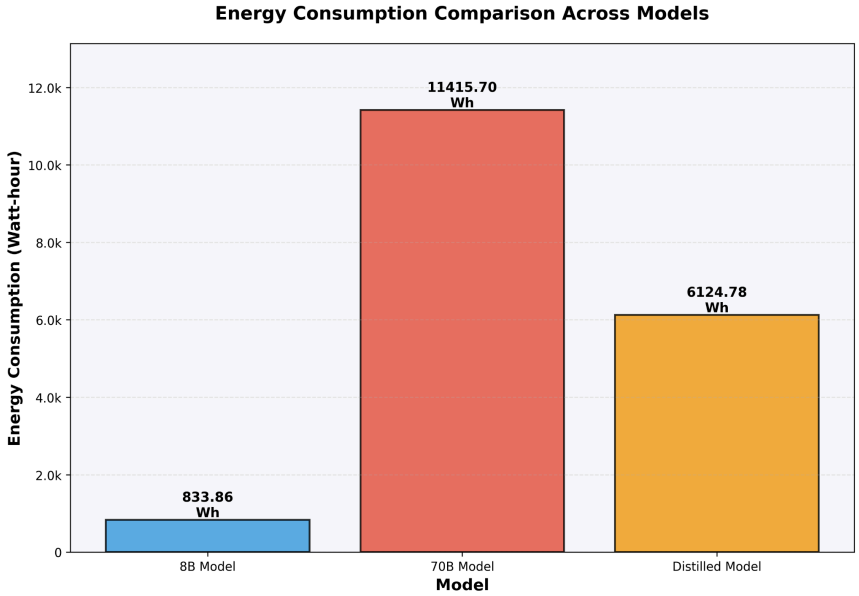


Fig 2. The above graph shows energy consumption across all 3 models with 70B having the highest consumption, followed by the distilled model, and lastly the 8B model

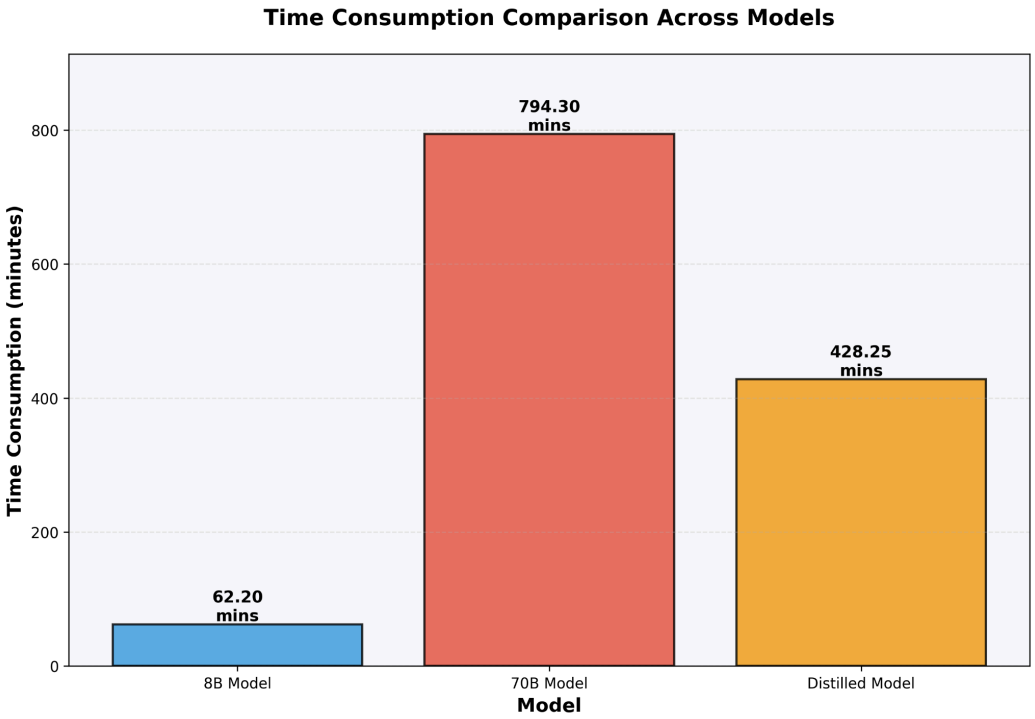


Fig 3. The above graph shows time consumption across all 3 models with 70B taking the longest, the distilled model cutting that time in half, and the 8B model performing the best.

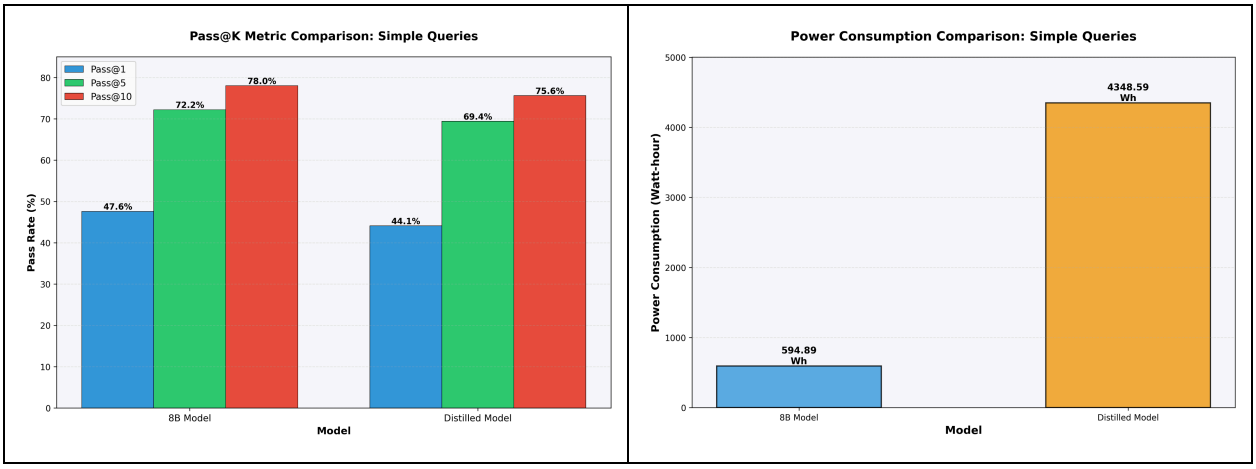


Fig 4. The two graphs above compare the pass@k performance and power consumption on simple queries between the 8B and distilled models

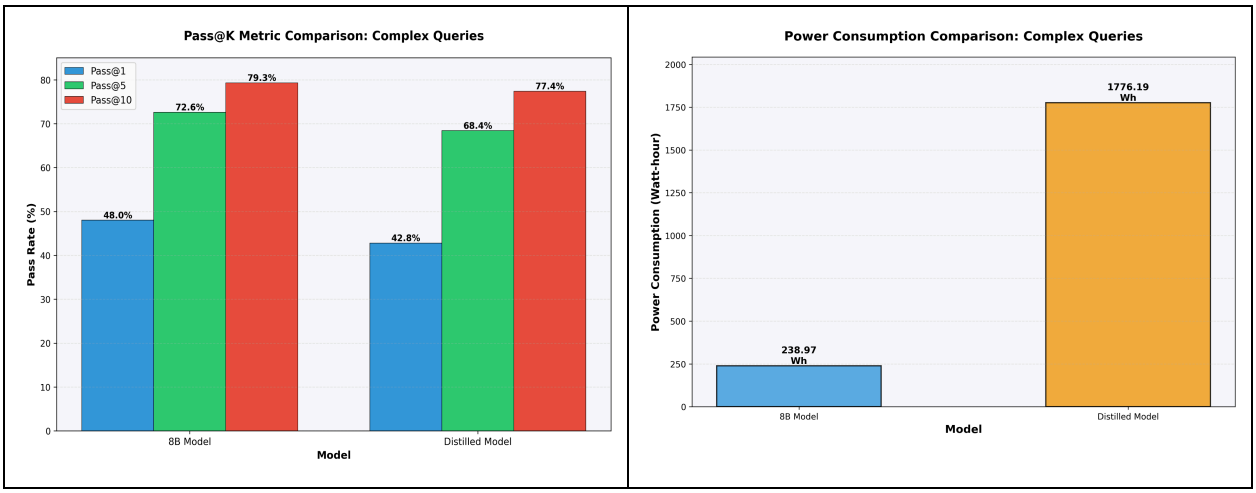


Fig 5. The two graphs above compare the pass@k performance and power consumption on complex queries between the 8B and distilled models

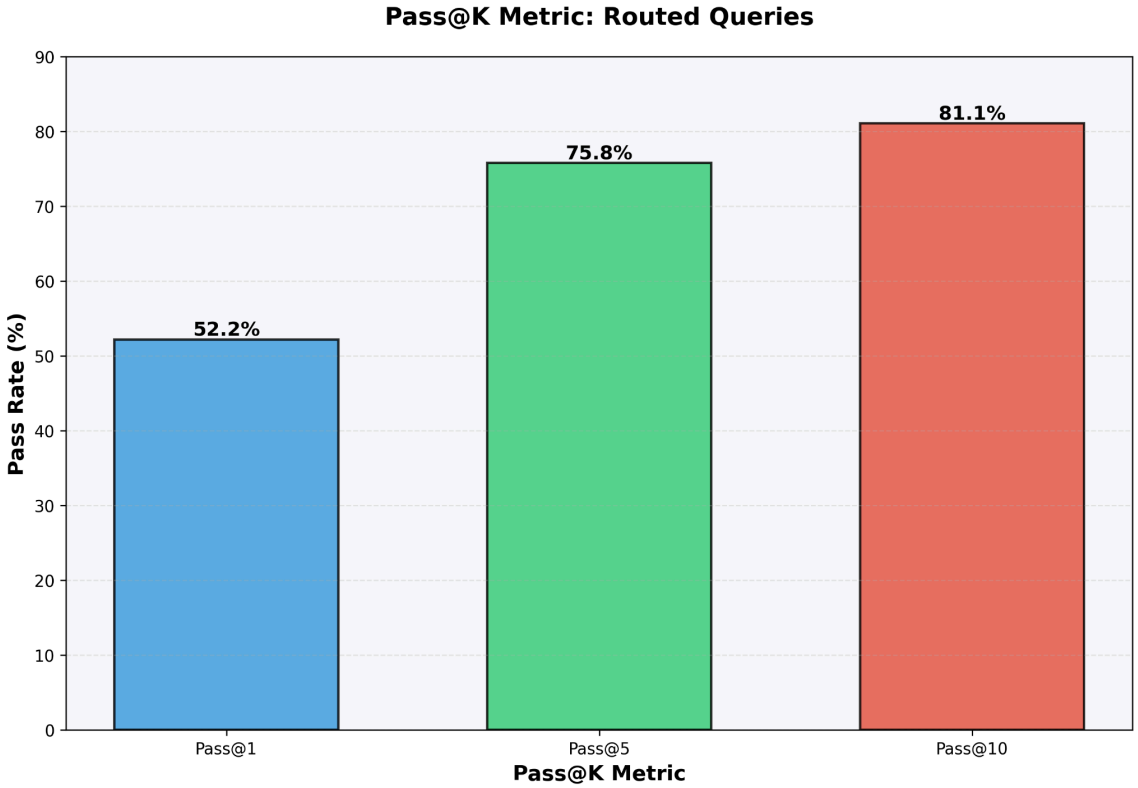


Fig 6. The above graph shows the pass rate on pass@k metrics for routed queries

Sources

The sources used for this project are not cited in the research itself as they were solely used to inform the decision to pursue knowledge distillation. All sources are provided to further inform the reader on decisions made on this project and the benefits of knowledge distillation in LLM compression.

Goyal, V., Khan, M., et al. (2024). *Enhancing Knowledge Distillation for LLMs with Response-Priming Prompting*. arXiv preprint https://arxiv.org/pdf/2412.17846.

Shirgaonkar, A., et al. (2024). *Knowledge Distillation Using Frontier Open-Source LLMs: Generalizability and the role of Synthetic Data*. arXiv preprint https://arxiv.org/pdf/2410.18588.

Timiryasov, I., & Tastet, J. (2023). *Baby Llama: Knowledge Distillation From an Ensemble of Teachers Trained on a Small Dataset With No Performance Penalty*. Association for Computational Linguistics https://aclanthology.org/2023.conll-babylm.24/.

Zhu, J., Li, M., Liu, J., Ma, S., & Wang, H. (2024). *A Survey on Model Compression for Large Language Models*. Association for Computational Linguistics https://aclanthology.org/2024.tacl-1.85/.

O'Neill, J., Dutta, S. (2023). *Self-Distilled Quantization: Achieving High Compression Rates in Transformer-Based Language Models*. Association for Computer Linguistics https://aclanthology.org/2023.acl-short.114.pdf.