Optimizing SLMs for Automated Email Responses: A Lightweight Fine-Tuned Model for Mobile Deployment

Suyash Srivastava[1], Juhi Srivastava[1], Kanishk Pachauri[1], Bharti Chugh[1]

[1]Department of Computer Science and Engineering, KIET Group Of Institutions, Delhi-NCR, Ghaziabad, 201206, Uttar Pradesh, India.

Contributing authors: suyashsrivastavam87@gmail.com ; srivastava.juhi.110@gmail.com; itskanishkp.py@gmail.com; bharti.kathpalia@gmail.com;

Abstract

Automating email responses with AI is useful, but running powerful models on mobile devices is challenging due to their size and computational demands. In this work, we distill knowledge from a large-scale model (GPT-4o) into a much smaller language model that can generate high-quality email responses while being lightweight enough for mobile deployment. We first use GPT-4o to create a high-quality supervised fine-tuning (SFT) dataset, capturing its reasoning and writing style. Then, we fine-tune a small model using this dataset, ensuring it learns to generate natural and context-aware responses efficiently. Our results show that this fine-tuned model performs well across different email scenarios while being fast and resource-efficient, making it a practical solution for automated email responses on mobile devices.

Keywords: LoRa (low-rank adaption technique), KG (knowledge distillation)

1 Introduction

Large Language Models (LLMs) have transformed the way we interact with artificial intelligence, powering everything from chatbots to content generation tools. These models, like GPT-4o, are incredibly powerful, capable of understanding and generating

human-like text with impressive accuracy. However, their sheer size and computational demands make them difficult to deploy in real-world applications that require speed and efficiency. As a result, there is a growing need for smaller, more efficient models that can perform well without the massive resource requirements of their larger counterparts.

This is where Small Language Models (SLMs) come in. These models, typically with fewer parameters than LLMs, strike a balance between performance and efficiency. While they may not match the raw power of larger models, they can be fine-tuned to achieve similar levels of accuracy for specific tasks. This makes them ideal for applications like customer support, chat-based assistants, and automated email responses—where quick, context-aware replies are essential but using a massive model is impractical.

Automated email responses are a great example of how AI is already making everyday communication easier. Services like Gmail's Smart Reply and Smart Compose help users respond to emails with just a tap, offering contextually relevant suggestions. These systems rely on machine learning models trained on vast amounts of email data to predict the best possible replies. While effective, they still depend on large-scale models running behind the scenes, which can introduce latency and require significant computational power. The challenge, then, is to create a lightweight model that can deliver high-quality responses without sacrificing speed or accuracy.

One way to achieve this is through knowledge distillation, a technique that allows a smaller model (the "student") to learn from a larger model (the "teacher"). Instead of training the smaller model from scratch, it absorbs knowledge from the larger model, learning to generate responses in a way that closely mimics the original. This approach has been widely used in NLP to compress large models while maintaining their effectiveness, making it a perfect fit for optimizing automated email response systems.

In this work, We tackle this challenge by first generating a high-quality dataset using GPT-4o and then fine-tuning a 1B parameter model using Supervised Fine-Tuning (SFT). By combining knowledge distillation with targeted fine-tuning, I aim to create a model that can generate accurate, context-aware email replies while being lightweight enough for real-world deployment. This approach ensures that we get the best of both worlds—high-quality responses without the heavy computational cost. .

2 Literature Review

2.1 Google Smart Reply

Automated email response generation has become an essential feature in modern communication tools, helping users respond quickly and efficiently. One of the most well-known implementations is Google's *Smart Reply* system, which provides users with short, contextually appropriate responses to emails. Introduced in Gmail's

Inbox, the system was responsible for generating nearly 10% of mobile email replies, demonstrating its effectiveness in real-world usage .

2.2 Smart Reply System Architecture

The Smart Reply system is based on a *sequence-to-sequence* (seq2seq) learning framework and utilizes *Long Short-Term Memory* (LSTM) networks to predict responses. Given an incoming email, the model generates a set of possible replies. Unlike traditional chatbot models, this system was designed with scalability, efficiency, and real-world applicability in mind . To ensure high-quality, relevant suggestions, the authors addressed several key challenges:

- **Response Quality**: Ensuring generated responses are grammatically correct and contextually appropriate.
- **Utility**: Maximizing the likelihood that at least one suggested response will be useful to the user.
- **Scalability**: Processing millions of emails per day with minimal latency.
- **Privacy**: Avoiding direct inspection of email contents while maintaining performance.

To tackle these issues, the Smart Reply system follows a multi-stage approach:

1. **Response Selection**: An LSTM-based model ranks potential responses based on their probability of being an appropriate reply.
2. **Response Set Generation**: A predefined set of responses is created using a semi-supervised graph-based clustering approach.
3. **Diversity Filtering**: Ensuring the displayed responses are not redundant and provide varied options.
4. **Triggering Model**: A lightweight neural network determines whether an email warrants Smart Reply suggestions.

2.3 Semantic Clustering and Response Generation

A notable innovation in the Smart Reply system is its use of a *graph-based clustering approach* to group responses. Instead of relying on manually curated responses, the system automatically clusters similar responses using the *Expander* framework. This enables the model to generate responses that are both diverse and contextually relevant.

Another important feature is the **diversity mechanism**, which ensures that responses cover different semantic intents. Instead of merely ranking the most probable responses, the system balances positive and negative replies to offer a range of choices. For example, if an email asks about availability, Smart Reply might offer a "Yes," a "No," and a neutral response.

2.4 Impact and Limitations

The deployment of Smart Reply demonstrated how deep learning can be applied effectively to streamline email communication. By providing quick, well-formed responses,

the system has significantly reduced the effort required for email replies, particularly on mobile devices. Moreover, its reliance on predefined response clusters helps maintain response quality and prevent inappropriate suggestions.

However, the approach does come with limitations:

- **Limited Personalization**: Responses are general and do not adapt to individual writing styles.
- **Fixed Response Set**: The use of predefined response clusters, while ensuring quality, limits the ability to generate more nuanced replies.
- **Scalability of LSTMs**: While effective, LSTMs can be computationally expensive, and more recent models such as transformers could provide improved contextual understanding with better efficiency.

While Smart Reply introduced an efficient, scalable approach to automated email responses, our research aims to build on this by leveraging GPT-4o for dataset generation and fine-tuning a 1B parameter model using Supervised Fine-Tuning (SFT). Unlike Smart Reply, which relies on predefined clusters, our approach seeks to fine-tune a small model to directly learn from a larger generative model through knowledge distillation. This allows the smaller model to retain the contextual depth of GPT-4o while being lightweight enough for real-time deployment.

3 Methodology

3.1 Dataset Generation

To create a high-quality dataset for training and evaluation, we used the FinePersonas-Conversations-Email-Summaries dataset, which consists of 344k emails and their corresponding summaries. The dataset generation process was conducted as follows:

- We randomly sampled **50k** email-summary pairs from the dataset as our base dataset.
- Each sampled email was provided as input to GPT-4o, prompting it to generate a response.
- The generated response was added to the evaluation prompt and again sent to Gpt-4o which evaluates the responses on relevance, conciseness, politeness and adaptability assigns score between 0-1 for each attribute.
- Responses meeting or exceeding a specified threshold were accepted into the dataset.
- If a response did not meet the threshold, feedback was incorporated based on the generated response and its score.
- An **iterative refinement process** was implemented:
  - If a response failed to meet the threshold, the model was provided feedback and allowed to regenerate the response up to three times.
  - If, after three iterations, the response still did not surpass the threshold, it was rejected.
- This process ensured that only **high-quality, contextually appropriate responses** were retained.

The final dataset, containing both the original emails and the **filtered, high-quality generated responses**, was used for further model training and fine-tuning. The sample data are uploaded here email-dataset . System prompts for data generation is provided in **Appendix 1**.

3.2 Scaling Data Generation

To efficiently scale the data generation process, we leveraged **semaphores** to manage **asynchronous requests** to **GPT-4o**, ensuring optimal utilization of API resources while maintaining system stability. The use of **asyncio.Semaphore** enabled controlled parallel processing, significantly **reducing the overall data generation time by up to 90%** compared to sequential execution.

3.2.1 Managing Concurrent Requests

One of the primary challenges in large-scale data generation is handling multiple API requests simultaneously without exceeding rate limits. To address this, we configured a **semaphore value of 50**, which allowed **up to 50 concurrent requests** at any given time. This configuration ensured that the system operated **efficiently within the constraints** of the API while **maximizing throughput**.

- The choice of **50 concurrent requests** was based on empirical testing, balancing speed and API compliance.
- This approach prevented excessive API throttling while ensuring a steady stream of data generation requests.
- The use of **asynchronous processing** allowed tasks to execute independently, minimizing idle time and enhancing system responsiveness.

3.2.2 API Constraint Compliance

To ensure uninterrupted data generation, we carefully adhered to the **token-per-minute** and **request-per-minute** limitations imposed by the API provider. The system was designed with built-in **rate-limiting mechanisms** to dynamically adjust request frequency based on real-time API responses.

- **Monitoring mechanisms** tracked the number of tokens and requests sent per minute.
- If the system approached the API-imposed limits, it would **temporarily pause** new requests to prevent exceeding constraints.
- This approach ensured **efficient quota utilization** while avoiding API rejections due to overuse.

3.2.3 Handling Failures with Exponential Backoff

API-based systems often encounter temporary failures due to **network issues, rate limits, or transient server errors**. To enhance robustness, we incorporated **exponential backoff retries**, which allowed the system to **gracefully recover from temporary failures** and continue processing without manual intervention.

- When an error occurred, the system would **wait for an exponentially increasing time interval** before retrying the request.
- The retry mechanism ensured that repeated failures did not overwhelm the API by sending too many rapid successive requests.
- This **adaptive retry strategy** significantly improved the reliability of the data generation pipeline by reducing request failures and minimizing downtime.

3.2.4 Performance Gains and Efficiency

By integrating **semaphores, API constraint management, and failure handling**, the system achieved **significant improvements in data generation efficiency**:

- **Up to 90% reduction in processing time**, compared to sequential execution.
- **Optimized concurrent request handling**, ensuring maximum throughput without exceeding API limits.
- **Automated failure recovery**, reducing manual intervention and improving system resilience.

This **optimized approach** to data generation enabled rapid and reliable large-scale processing, making it well-suited for high-volume applications that require continuous, high-quality text generation.

3.3 Training Details

3.3.1 Model and LoRA Configuration

We fine-tuned the Meta LLaMA-3.2-1B-Instruct model using Supervised Fine-Tuning (SFT) with Low-Rank Adaptation (LoRA) to improve efficiency in parameter tuning. LoRA is a method that enables efficient fine-tuning of large-scale models by introducing trainable low-rank matrices into the attention layers while keeping the majority of pre-trained model parameters frozen. This significantly reduces memory and computational overhead while maintaining strong adaptation capabilities.

The following configuration was used during fine-tuning:

- **Base Model:** Meta LLaMA-3.2-1B-Instruct
- **LoRA Configuration:**
  - **Rank (r) = 16:** This determines the dimensionality of the low-rank decomposition. A higher rank allows for more expressive adaptation but increases computational cost.
  - **LoRA Alpha = 16:** A scaling factor applied to LoRA updates, which helps control the magnitude of weight updates.
  - **LoRA Dropout = 0:** No dropout was applied, ensuring all parameters contribute to adaptation.
- **Target Modules:**
  - `q_proj`, `k_proj`, `v_proj`: These correspond to the query, key, and value projection layers in the self-attention mechanism.

  - `up_proj`, `down_proj`: These are part of the feedforward network (FFN) layers, responsible for transforming intermediate representations.
  - `o_proj`: The output projection layer, which aggregates attention heads.
  - `gate_proj`: Helps regulate information flow through gating mechanisms.

- **Efficient Implementation:**
  - We used `rslora`, an advanced implementation of LoRA optimized for better memory efficiency and computational speed. This reduces GPU memory consumption while retaining adaptation performance, making it more suitable for resource-constrained environments.

By leveraging LoRA, we significantly reduced the number of trainable parameters while still achieving effective fine-tuning results.

3.3.2 Training Hyperparameters

Fine-tuning was conducted using a carefully selected set of hyperparameters to balance convergence speed and stability:

- **Batch Size:** 32 – A moderate batch size was chosen to optimize memory utilization while maintaining training stability.
- **Epochs:** 2 – Although fewer epochs were used, the pre-trained foundation of LLaMA-3.2-1B-Instruct allowed rapid adaptation with a relatively small dataset.
- **Learning Rate:** 5×10^{-5} – This learning rate was selected through empirical testing, ensuring a balance between fast convergence and stability.
- **Gradient Accumulation Steps:** 2 – Used to effectively increase the batch size while keeping GPU memory requirements manageable.
- **Warmup Steps:** 100 – A warmup period helps stabilize early training by gradually increasing the learning rate from zero to its peak value, preventing abrupt weight updates.
- **Max Gradient Norm:** 2 – Gradient clipping was applied to prevent exploding gradients, which is particularly important when fine-tuning large models.
- **Precision:** Mixed-Precision Training using bfloat16/float16 – This allows efficient computation with minimal precision loss, significantly reducing memory consumption without affecting model accuracy.

Training with mixed precision (bfloat16 or float16) leverages GPU hardware optimizations, enabling faster computation while maintaining numerical stability.

3.3.3 Optimization and Checkpointing

To ensure robust training, we used AdamW, an adaptive optimization algorithm that combines the benefits of Adam with weight decay regularization to prevent overfitting.

- **Optimizer:** AdamW – Selected for its ability to adapt learning rates for each parameter, providing stable and efficient convergence.
- **Checkpointing:**

– Model checkpoints were saved every 100 steps to ensure progress tracking and safeguard against potential training interruptions.
– The best-performing model (based on validation loss) was retained for evaluation.
– Intermediate checkpoints were stored for potential model reloading and fine-tuning adjustments.

The combination of regular checkpointing and AdamW optimization facilitated a controlled and efficient training process, reducing the risk of catastrophic forgetting and ensuring reproducibility.

3.3.4 Dataset and Preprocessing

The dataset used for fine-tuning was curated to align with the task of email-response generation, a crucial application in natural language processing (NLP). The dataset consisted of email-response pairs, where each email serves as an input, and the model generates the corresponding response.

- **Data Composition:**
  – The dataset was sourced from a diverse range of formal and informal email exchanges to improve model generalization.
  – Preprocessing steps ensured grammatical correctness, removing noisy or ambiguous samples that could degrade performance.
- **Data Split:**
  – 95% Train / 5% Test – The majority of the data was used for training, while a small portion was reserved for evaluating generalization performance.
- **Tokenization:**
  – A chat template was applied to format input-output pairs consistently.
  – Sequences were truncated to a maximum length of 1024 tokens to fit within the model's input capacity while retaining essential context.

Fine-tuning was conducted using `torch.bfloat16` or `torch.float16`, depending on hardware support. This ensured efficient computation without precision loss, making training feasible on GPUs with limited memory.

To track and analyze training progress, Weights & Biases (W&B) was integrated for experiment tracking, hyperparameter tuning, and visualization. This provided insights into loss curves, gradient updates, and model performance across training epochs.

4 Results after finetuning

After completing the fine-tuning process, we evaluated our model on a carefully curated test set consisting of 5,000 samples. The goal of this evaluation was to assess the quality, coherence, and semantic accuracy of the generated responses. To achieve this, we employed two widely recognized evaluation metrics: BERTScore and BLEU Score, each providing unique insights into different aspects of text generation quality.

4.1 BERTScore

BERTScore is a sophisticated evaluation metric that measures the semantic similarity between the generated responses and the reference text. Unlike traditional n-gram-based approaches, BERTScore leverages contextual embeddings from a pre-trained transformer model (such as BERT) to compare the meaning of words rather than just their lexical overlap.

- The metric computes the cosine similarity between token embeddings in the generated text and the reference text, ensuring a deeper understanding of contextual meaning.
- This approach is particularly beneficial for evaluating open-ended text generation, where strict word-for-word matching may not always capture the quality of a response.
- A higher BERTScore indicates that the generated responses are semantically accurate and align closely with human-written text.

4.2 BLEU Score

The BLEU (Bilingual Evaluation Understudy) Score is one of the most established metrics for evaluating text generation models, especially in machine translation and text summarization tasks. Unlike BERTScore, which focuses on meaning, BLEU evaluates the n-gram overlap between the generated response and the reference text.

- BLEU computes precision scores for unigrams, bigrams, trigrams, and higher-order n-grams, providing a quantitative measure of textual similarity.
- The score is particularly useful for assessing how closely the generated text follows the structure of the reference text while maintaining fluency and grammatical correctness.
- While BLEU is effective for structured text generation, it may not fully capture semantic nuances, which is why it is often combined with other metrics like BERTScore.

4.3 Final Performance

After fine-tuning, our model achieved strong performance on the test set, as demonstrated by the following evaluation scores:

- BERTScore: 0.832
- BLEU Score: 64.3

These results indicate that the fine-tuned model generates responses that are both semantically meaningful and structurally well-formed. The high BERTScore suggests that the model maintains contextual accuracy, while the strong BLEU score reflects a high degree of textual alignment with human-written responses.

Fig. 1 Train-loss vs Epoch

4.4 Analysis and Observations

- The high BERTScore confirms that the model effectively understands and retains the meaning of input queries, producing coherent and contextually relevant responses.
- The BLEU score of 64.3 demonstrates that the model maintains linguistic consistency and fluency, with a significant overlap between generated and reference texts.
- The combination of these metrics suggests that the fine-tuning process successfully enhanced the model's ability to generate natural, high-quality responses.
- Further improvements could be achieved by incorporating human evaluation metrics, such as ROUGE for summarization tasks or GPT-based evaluation for more nuanced assessments.

4.5 Future Considerations

While the results indicate a high level of performance, future work could focus on the following areas for further refinement:

- Expanding the dataset to include more diverse writing styles and domains to improve generalization.
- Fine-tuning on additional linguistic patterns, such as informal or domain-specific emails, to enhance adaptability.
- Exploring reinforcement learning approaches, such as Reinforcement Learning with Human Feedback (RLHF), to optimize response generation based on user preferences.

By addressing these aspects, we can further improve the model's ability to generate human-like, contextually relevant, and highly accurate responses across a wide range of applications.

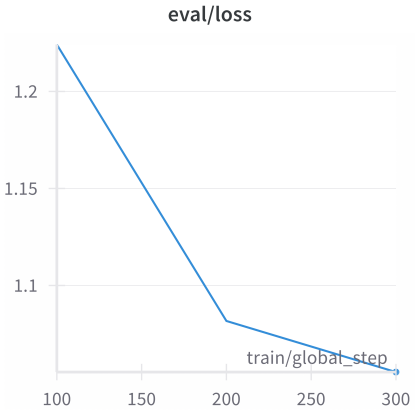

Fig. 2 Eval-loss vs Epoch

5 Usage and deployment

To run LLaMA 3.2 1B on an Android device, follow these steps:

1. **Download and Set Up Torchchat Framework**: Download the Torchchat framework, which includes a demo Android app for running LLaMA models. Ensure you have the necessary '.aar' file for the Java library and JNI library.
2. **Prepare the Android Project**: Open the demo project in Android Studio, located at 'torchchat/edge/android/torchchat'. Trust the project and wait for Android Studio to complete the initial setup.
3. **Copy Model Files to Device**: Connect your Android device to your computer via USB. Use `adb` commands to create a directory on the device (`local/tmp/llama`) and copy the `llama3_2-1b.pte` model file and the tokenizer file to this directory.
4. **Run the App on the Device**: Run the demo app on your connected Android device using Android Studio. Interact with LLaMA 3.2 1B through the app's interface by entering prompts and generating responses.
5. **Considerations and Limitations**: Be aware of potential limitations in the demo app, such as response formatting issues and token limits. For better performance, consider using optimized frameworks like XNNPACK for acceleration.

These steps are derived from this article.

References

[1] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukacs, M. Ganea, P. Young, and V. Ramavajjala, "Smart reply: Automated response suggestion for email," *arXiv preprint arXiv:1606.04870 [cs.CL]*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1606.04870

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762 [cs.CL]*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.03762

[3] [Zhenyan Lu , Xiang Li , Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang , Nicholas D. Lane, Mengwei Xu]. *[SMALL LANGUAGE MODELS: SURVEY, MEASUREMENTS, AND INSIGHTS]*. arXiv:2409.15790, 2024. https://doi.org/10.48550/arXiv.2409.15790

[4] [Aldo Pareja , Nikhil Shivakumar Nayak , Hao Wang , Krishnateja Killamsetty3 , Shivchander Sudalairaj , Wenlong Zhao1, , Seungwook Han , Abhishek Bhandwaldar , Guangxuan Xu , Kai Xu , Ligong Han , Luke Inglis , Akash Srivastava]. *[UNVEILING THE SECRET RECIPE: A GUIDE FOR SUPERVISED FINE-TUNING SMALL LLMS]*. arXiv:2412.13337, 2024. https://doi.org/10.48550/arXiv.2412.13337

[5] [Edward Hu, Yelong Shen ,Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li Shean, Wang Lu Wang Weizhu Chen]. *[LORA: LOW-RANK ADAPTATION OF LARGE LAN- GUAGE MODELS]*. arXiv:2106.09685, 2021. https://doi.org/10.48550/arXiv.2106.09685

[6] Rohit Chakraborty, Ivan Kobyzev, Ilya Grishanov, Xanh Ho, Mehdi Rezagholizadeh, Pascal Poupart. *Efficient Compression of Deep Learning Models by Structured Matrices for Knowledge Distillation with Small Student Models*. arXiv:2402.13116, 2024. https://doi.org/10.48550/arXiv.2402.13116

[7] Ashish Kumar Patra, Chang Gao, Anjali Shenoy, Nadav Cohen, Nadav Dym. *A Neural PDE Solver with Temporal and Spatial Adaptivity*. arXiv:2306.08543, 2023. https://doi.org/10.48550/arXiv.2306.08543

[8] Sahibsingh A. Dudani, Simon W. Chang, Nainoa Avina, Alex Kemp, Atiksh Bhardwaj, Wendy M. K. Tam, Sayan Ghosh, Samir Yitzhak Gadre, Sameer Singh, Yuxin Chen, Reza Takapoui, Yang Yuan. *FlashTune: Extended Context Inference for Autoregressive Language Models*. arXiv:2410.22182, 2024. https://doi.org/10.48550/arXiv.2410.22182

[9] Juan F. Zamora, Rafael Cámara, Jesús Urías, Rafael Salas. *Exploiting GPT for synthetic data generation: An empirical study*. arXiv:2404.08259, 2024. https://doi.org/10.48550/arXiv.2404.08259

[10] Junhan Wang, Yi-Chen Li, Jinxing Li, Xiaoyan Wang, Xin Jiang, Wei Yuan. *GPT in Research: A Systematic Review on Data Augmentation*. arXiv:2406.01743, 2024. https://doi.org/10.48550/arXiv.2406.01743

[11] Xiangru Tang, Yuzuru Okajima, Armin W. Thomas, Jonathan W. Pillow. *Prompt Retrieval for Large Language Models*. arXiv:2402.12354, 2024. https://doi.org/10.48550/arXiv.2402.12354

[12] Yuriy Koshulko, Olena Maksymova. *Review of the Main Approaches to Automated Email Answering*. International Journal of Information Technology and Computer Science, Vol. 8, No. 8, pp. 26-32, 2016. https://doi.org/10.5815/ijitcs.2016.08.04

[13] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, Shen Li. *PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel*. Proceedings of the VLDB Endowment, Vol. 16, No. 12, pp. 3848-3860, 2023. https://doi.org/10.14778/3611540.3611569

[14] Chuanpeng Yang, Wang Lu, Yao Zhu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, Yiqiang Chen. *Survey on Knowledge Distillation for Large Language Models: Methods, Evaluation, and Application*. arXiv preprint, 2024. https://arxiv.org/abs/2407.01885v1

[15] Alexander Wettig, Anian Ruoss, Martin Maas, Harsha Nori, Thomas Steinke, Misha Belkin, Yao Zhao. *ZigZag Training: Improving Convergence, Preventing Collapse and Runtime Instability in Large Language Models*. arXiv:2412.19437, 2024. https://doi.org/10.48550/arXiv.2412.19437

[16] [Karmvir Singh Phogat, Sai Akhil Puranam, Sridhar Dasaratha, Chetan Harsha, Shashishekar Ramakrishna]. *[Fine-tuning Smaller Language Models for Question Answering over Financial Documents]*. arXiv:2408.12337, 2024. https://doi.org/10.48550/arXiv.2408.12337

A Appendix

A.1 System Prompts

Generation Type	System Prompt
Response Generation	[You are trained to generate professional, polite, and concise email replies. Given an email, generate a short and relevant response that acknowledges the content while keeping it general and adaptable to different contexts. Ensure the tone is neutral, courteous, and professional. Instructions 1. Keep the response between 1 to 3 sentences. 2. If clarification is needed, express willingness to assist further. 3. Avoid unnecessary details while ensuring politeness. 4. Do not include subject in response only include body of email response that you are generating. 4. Use placeholders like "name" and "specific detail" where customization is expected. 5. Return the reponse in clear json format. email: email]
Response Evaluation	[You are an AI email response evaluator. Your task is to assess email responses based on four key criteria: relevance, conciseness, adaptability, and politeness. Assign a score between 0 and 1 for each criterion and provide actionable feedback for improvement. **Evaluation Criteria:** 1. **Relevance (0-1):** Measures how well the response addresses the main points of the email. 2. **Conciseness (0-1):** Evaluates if the response is clear and to the point without unnecessary details. 3. **Adaptability (0-1):** Assesses how well the response is tailored to the recipient's tone, context, and needs. 4. **Politeness (0-1):** Checks if the response maintains a professional and courteous tone. **Output Format:** You must return a **valid JSON object** structured as follows: "'json "relevance": 0.85, "conciseness": 0.90, "adaptability": 0.75, "politeness": 1.00, "feedback": "relevance": "The response covers most key points but could address the sender's specific concern more directly.", "conciseness": "The response is clear, but a minor reduction in wordiness would improve readability.", "adaptability": "Consider adjusting the tone slightly to better align with the sender's formality level.", "politeness": "The response is perfectly polite."]
Response Evolution	[You are an AI trained to improve email replies based on evaluation feedback. Given an email, an initial response, an average score, and feedback, your task is to rewrite the reply to enhance clarity, relevance, politeness, and adaptability while keeping it concise. Instructions: 1. Analyze the given input: Identify areas for improvement based on the feedback. 2. Rewrite the response to address the feedback while ensuring: - Relevance: Fully responds to the email's intent. - Conciseness: Brief and to the point. - Politeness: Uses a professional and courteous tone. - Adaptability: Includes placeholders (e.g., "name") for easy customization. 3. Preserve the original meaning but refine structure, tone, and clarity. Input Format: Email: "email" Generated Response: "generated$_r esponse$" $AverageScore$: $scoreFeedback$: "$feedback$" Expected Output Format: Refined Response:]

Table 1 System Prompts for Response Generation, Evaluation, and Evolution.