Comprehensive Technical Documentation (Email Generation Model)

A. Overview

This project, "SMARTGENLEAD," is an AI-powered tool designed to solve a critical bottleneck in B2B sales: writing personalized cold outreach emails at scale. The system reads a structured JSON profile of a sales lead—including their career history, recent activity, and company updates—and generates a short, polite, and highly relevant cold email that is ready to send.

The system is a prompt-based inference pipeline using a pre-trained Large Language Model (LLM).

The data flows as follows:

1. **Input:** A user_json object containing objective (the sales goal) and profile_data (the lead's information).
2. **Prompt Engineering (Python):** The raw JSON is processed by a Python script (build_lead_profile, mbti_to_style_profile) to create two clean context blocks: lead_profile and style_profile.
3. **Prompt Assembly:** These context blocks are inserted into a user_prompt and combined with a system_prompt that defines the AI's rules and required JSON output format.
4. **Model Inference:** The complete prompt is sent to the mistralai/Mistral-7B-Instruct-v0.2 model loaded in 4-bit precision.
5. **Output Parsing:** The model's raw string output (which should be a JSON object) is parsed.
6. **Post-processing:** The subject and body are extracted. Placeholders like {first_name} are filled, and a compliance footer is added.
7. **Final Output:** A ready-to-send email with a subject and body.

The project is deployed as a Kaggle Demo Notebook.

- **Backend Model:** mistralai/Mistral-7B-Instruct-v0.2 is loaded directly into the notebook's GPU runtime from Hugging Face.

B. Technical Documentation

Section	Description
1. Environment Setup	**Python Version:** 3.10+ **Hardware:** GPU required (Kaggle T4 or Colab T4). **Key Dependencies:** transformers, torch, accelerate, bitsandbytes, sentencepiece. **Install:** pip install transformers torch accelerate bitsandbytes sentencepiece
2. Data Pipeline	**Source:** The project uses synthetically generated, structured JSON data representing sales leads. No external datasets are used for training. **Preprocessing:** The normalize_profile function ensures all expected keys (name, headline, career_history, etc.) exist in the input dictionary to prevent runtime errors. The build_lead_profile function selects and truncates data (e.g., top 8 skills, top 3 recent activities) to create a concise profile that fits within the model's context window.

3. Model Architecture	**Model:** mistralai/Mistral-7B-Instruct-v0.2. This is a 7.3-billion parameter, pre-trained, instruction-following Transformer model. **Reasoning:** Chosen for its strong performance in following complex instructions and adhering to strict JSON output formats, which was a failure point for smaller models like TinyLlama-1.1B. **Quantization:** The model is loaded using bitsandbytes in 4-bit precision (nf4) with torch.float16 compute data type. This drastically reduces memory usage from ~30GB to ~5GB, making it runnable on free-tier Kaggle/Colab GPUs.
4. Training Summary	This project involves fine-tuning. Success is achieved through prompt engineering, selecting a capable base model, and efficiently generating emails based on the enron dataset

5. Evaluation Summary	**Method:** Evaluation was performed in Milestone_5_LLM.ipynb using a rule-based evaluation script. This script checks 7 test leads against a set of "hard rules" from the system prompt. **Key Metrics (TinyLlama 1.1B):** The initial model (TinyLlama-1.1B-Chat-v1.0) **failed significantly**, scoring an **average quality of 4.86/10**. * word_count_in_range_rate: **0%** (all emails were too long or too short) * greeting_ok_rate: **0%** (failed to use the lead's name) * closing_ok_rate: **0%** (failed to use the sender's name) * personalization_ok_rate: **42.9%** **Insight:** The model was not following instructions, outputting templates instead of executing the task. **Solution:** Switching to Mistral-7B resolves these issues, as it correctly follows the prompt, uses the lead's information, and formats the output as JSON.

6. Inference Pipeline	The core inference pipeline is the generate_email function. 1. It takes user_json, sender_name, and address. 2. It calls build_lead_profile and mbti_to_style_profile to create the prompt context. 3. It assembles the messages list with role: system and role: user. 4. It calls the chat() function. 5. chat() applies the tokenizer's chat template, generates tokens, and decodes the response. 6. generate_email then parses the returned JSON string (with regex-based fallback) and populates the final email body. **Code Snippet:** email = generate_email(user_json_demo, ...)
7. Deployment Details	**Platform:** Kaggle Notebook. **Model Hosting:** The model is downloaded from Hugging Face Hub (mistralai/Mistral-7B-Instruct-v0.2) and loaded into the notebook's runtime memory. **Interaction:** A user runs the final cell in the notebook. The cell contains a demo user_json_demo variable and calls the

	generate_email function. The final subject and body are printed to stdout.
8. System Design	**Modularity:** The system is highly modular. The prompt (SYS_PROMPT), style logic (mbti_to_style_profile), and model (MODEL_ID) can all be changed independently. **Data Flow:** The use of structured JSON as the "ground truth" for the model is a key design choice, making the system auditable and controllable. **Robustness:** The generate_email function includes a try...except block for JSON parsing and a regex-based fallback. If the model fails to produce perfect JSON, the system attempts to extract the content anyway, preventing a hard crash.
9. Error Handling	**Model Failure (Bad JSON):** Handled by the try...except block and regex fallback in generate_email. If parsing fails, a warning is printed, and the raw model output is used as the email body. **Input Failure (Missing Keys):** Handled by the normalize_profile function, which uses .setdefault() to ensure all required keys exist, preventing KeyError exceptions.

10. Reproducibility	**Model:** mistralai/Mistral-7B-Instruct-v0.2 **Config:** 4-bit quantization (see section 3). **Code:** The provided Kaggle notebook is self-contained. **Seeds:** Generation temperature is set to 0.6 for creative but relatively stable outputs. For full reproducibility, do_sample=False or setting a global torch.manual_seed() would be required (not implemented, as variance is desired).

C. User Documentation (for Non-Technical Users)

This Kaggle notebook is a demo of "SMARTGENLEAD," a tool that automatically writes personalized B2B cold emails. You provide information about a person (a "lead"), and the AI writes a short, professional email to them, mentioning their specific job, skills, and recent company news.

- How to Use This Notebook

Follow these steps to generate an email:

1. **Run the Setup Cells:** In the notebook, run the first few cells (labeled "1. SETUP", "2. LOAD MODEL", "3. DEFINE INFERENCE FUNCTION", etc.). This will install the libraries and load the AI model. This may take a few minutes.
2. **Find the Demo Cell:** Scroll to the bottom of the notebook to the final cell (labeled "6. DEMO: Run the pipeline").

(Optional) Edit the Lead: You can edit the user_json_demo variable to change the lead's information. For example, you can change the name or company:

```
Python
user_json_demo = {
  "objective": "My new sales goal...",
  "profile_data": {
    "name": "Jane Doe",
    "headline": "CEO at Example Inc",
    ...
  }
}
```

3.
4. **Run the Demo Cell:** Click the "Run" button on this final cell.
5. **View the Output:** The notebook will print the AI's raw thoughts (labeled "RAW MODEL OUTPUT") and then show the final, clean email at the very bottom, formatted with a "SUBJECT" and "BODY".

Troubleshooting

- **"CUDA out of memory":** This means the GPU ran out of memory. Try restarting the notebook: in the menu, go to **Runtime > Restart session**. Then run the cells one by one again.
- **"Model did not return a JSON object":** You may see this warning. It means the AI's output was slightly malformed, but the script likely recovered. The final email should still be usable.
- **Email is irrelevant:** The AI's quality depends on the input. Ensure the objective is clear and the profile_data is accurate.

D. API Documentation

This project is deployed as a demo notebook, not a REST API. Therefore, this section is not applicable. The core function generate_email(user_json, ...) acts as the internal API for the notebook.

E. Licensing & Dataset References

- **Code License:** The Python code in this notebook is licensed under the **MIT License**.
- **Model License:** The base model, mistralai/Mistral-7B-Instruct-v0.2, is licensed under the **Apache 2.0 License**.
- **Dataset License:** The data used for demonstration (user_json_...) is synthetic and proprietary to this project. It is not licensed for reuse.

F. Future Work / Maintenance Notes

- **Known Limitations:** The system relies on a 7B parameter model. While far better than 1.1B, it can still occasionally fail to follow the JSON format or word count. The 4-bit quantization also slightly reduces accuracy.
- **Possible Extensions:**
    - **RAG Pipeline:** Instead of passing all data in one JSON, the system could be rebuilt as a RAG pipeline. The user's objective would be used to *query* a vector database containing the lead's information, retrieving only the *most relevant* facts to include in the email.
    - **Web API:** The notebook logic could be wrapped in a **FastAPI** or **Flask** server and deployed on a service like **Hugging Face Spaces** or **Render** to provide a live API endpoint.