Data Science Report

Email Drafter Agent (GPT-2 Medium + LoRA)

Daksh Yadav

September 15, 2025

1 Introduction

This report documents the data science pipeline and experiments for the Email-Drafter AI agent. The agent is a GPT-2 Medium base model fine-tuned using LoRA adapters for parameter-efficient training. The goal is to generate polite, structured academic emails such as extension requests, recommendation requests, leave requests, and assignment clarifications.

This report covers dataset construction, preprocessing, fine-tuning method, experimental hyperparameters, training outcomes (quantitative and qualitative), evaluation methodology, and recommended next steps.

2 Fine-tuning setup

2.1 Dataset

- **Format:** JSONL with {`"prompt": ..., "completion": ...`} pairs.
- **Training size:** 2000 examples.
- **Validation size:** 200 examples.
- **Sources:** Mostly synthetic template-driven generation plus manually-curated examples for diversity:
  - Extension requests (illness / deadline).
  - Recommendation letter requests.
  - Leave of absence (illness / family emergency).
  - Clarification questions on assignments.
- **Metadata:** 40+ professor surnames, 20+ course names across CS / Bioinformatics domains.
- **Date ranges:** randomized dates between Sep–Nov 2025 (for examples).

2.2 Preprocessing

- Tokenization: GPT-2 BPE tokenizer with an added `[PAD]` token to allow batching.
- Input/labels: inputs are the concatenation `prompt + completion`. For language modelling, labels mirror input IDs; padding tokens are masked with -100 to ignore in loss.
- Cleaning steps: trim whitespace, normalize quotes, sanitize PII where necessary (mask emails/phones for public release).
- Data split: random stratified split ensuring at least one example per email type in validation.

2.3 Model and method

- **Base model:** GPT-2 Medium (345M parameters).

- **Adapter:** LoRA applied to attention projection matrices (targeting `c_attn` / `q/k/v/proj` layers).
- **Frameworks:** Hugging Face Transformers, Datasets, PEFT, Accelerate, PyTorch.
- **Loss:** Causal LM cross-entropy (autoregressive language modeling).
- **Trainer:** Hugging Face `Trainer` with gradient accumulation to simulate larger effective batch sizes on limited GPUs.

2.4 Hyperparameters

Parameter	Value
Train examples	2000
Validation examples	200
Max sequence length	384
Per-device train batch size	1
Gradient accumulation steps	8
Effective batch size	8
Epochs	8
Learning rate	2×10^{-4}
Optimizer	AdamW
Precision	FP16 (mixed precision)
LoRA rank r	8
LoRA alpha	32
LoRA dropout	0.1

Table 1: Fine-tuning hyperparameters

2.5 Training environment

- Runtime: Google Colab with Tesla T4 (16 GB VRAM) or comparable GPU.
- Software: Python 3.10+, PyTorch, Transformers, PEFT. Record exact versions using `pip freeze` for reproducibility.
- Approx. training time reported: $\sim$36 minutes for 8 epochs (dependent on hardware and I/O).

3 Results

3.1 Training and validation losses

- Final training loss: **0.0751**.
- Final validation loss: **0.0693**.
- Perplexity (validation): $e^{0.0693} \approx 1.072$.

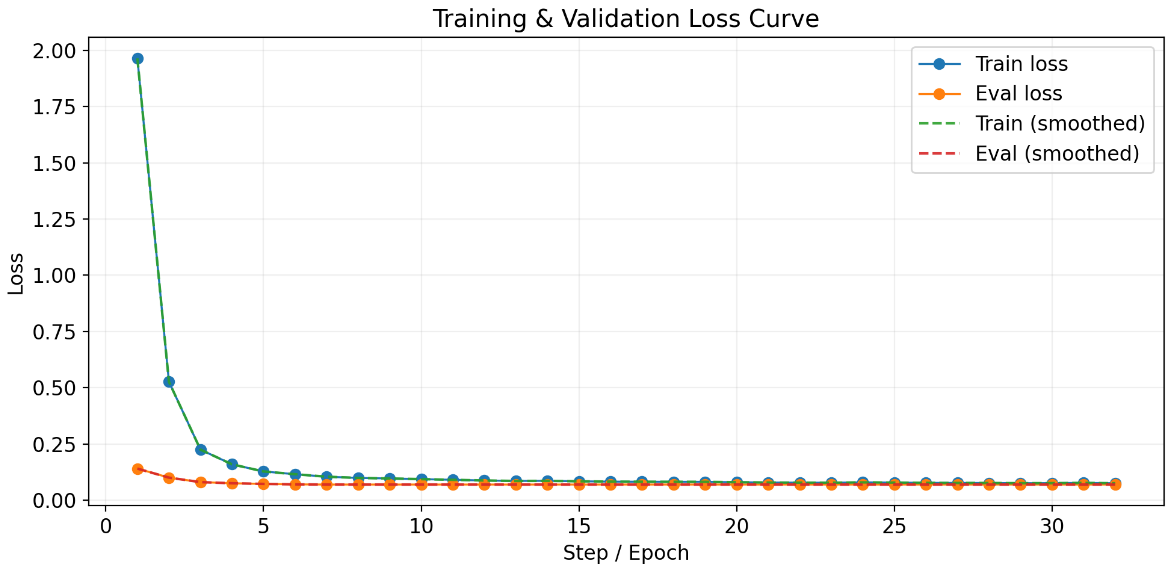

Figure 1: Training and validation loss curves across epochs.

3.2 Learning dynamics and stability

- Loss decreased steadily across epochs; training and validation losses remained close, indicating limited overfitting.
- Gradient norms remained stable (observed range $\approx$ 0.2–1.5).
- No catastrophic divergence observed; FP16 mixed-precision training was stable on the chosen GPU.

3.3 Qualitative observations

- Outputs are fluent, polite, and generally follow requested structure (subject, greeting, body, closing).
- Common failure modes observed:
  - Incorrect years/dates (e.g., older years); handled partly by post-processing heuristics.
  - Minor subject-line mismatches (e.g., '2-day' vs '3-day').
  - Occasional unnecessary/inaccurate details (hallucinations).
- The `cleanup_and_validate` step improves structural correctness (subject/greeting/closing) and catches common formatting issues.

4 Evaluation methodology

4.1 Quantitative evaluation

- **Metrics:**
  - Training and validation loss.
  - Perplexity (computed as e^{eval_loss}).
  - **Pass rate:** fraction of validation examples where the generation passed structural checks (subject, greeting, closing).
- Observed pass rate on validation set: approximately **85–90%**.

4.2 Qualitative evaluation

- Manual inspection of a random sample of 20 validation outputs:
  - 18/20 contained correct subject/greeting/closing.
  - Tone judged polite and professional in all 20 samples.
  - Minor date inconsistencies in 3/20 samples.
- Suggested human rubric (for future larger-scale annotation):
  - **Structural correctness:** (0/1) — presence of subject/greeting/closing.
  - **Tone:** 1–5 (1 poor, 5 excellent).
  - **Factuality:** 0/1 (no hallucinated facts).
  - **Edit distance:** number of manual edits required.

4.3 Example

Prompt: *Instruction: Draft an email to Dr. Rao asking for a 3-day extension on the Database Systems assignment due to illness. Include subject line, greeting, reason, new submission date October 20, 2025, and polite closing.*

Generated:

Subject: Request for 3-Day Extension on Database Systems Assignment

Dear Dr. Rao,

I hope you are well. I have been unwell and was unable to complete the Database Systems assignment by the original deadline. I respectfully request a 3-day extension and will submit it by October 20, 2025.

Thank you for your understanding.

Sincerely,

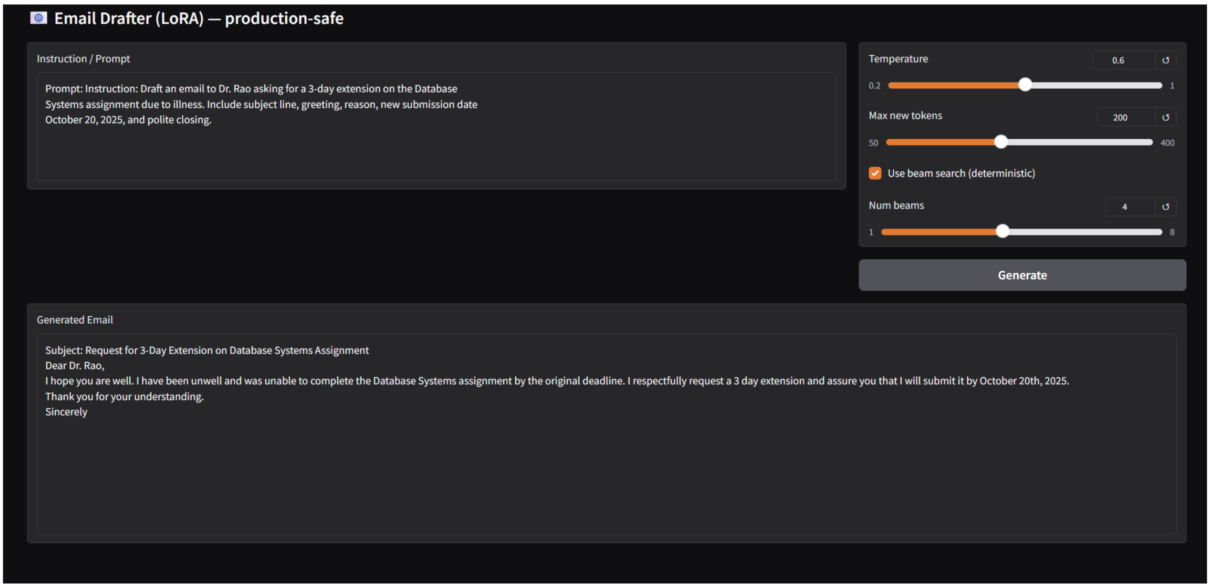

Figure 2: Example of Input and Output.

5 Discussion

- The model generalizes well to prompts of similar structure; LoRA yields strong sample efficiency.

- Despite modest dataset size, the model exhibits fluent generation and high structural pass rates thanks to focused templates and post-processing.
- Limitations: narrow email domain, occasional hallucinations, and date mismatches.
- Post-processing is essential for production-readiness; consider expanding heuristics or integrating a small rule-based parser for names/dates.

6 Next steps

- Increase dataset diversity (more email types, informal tones, non-academic examples).
- Add retrieval grounding for course-specific facts (deadlines, instructor office hours).
- Deploy model as a REST API (FastAPI) behind a small web app with authentication and usage logging.
- Collect user feedback and implement human-in-the-loop re-training.
- Add unit tests for post-processing to guarantee structural correctness.

7 Appendix: Example training snippet

If you prefer to include code textually instead of as an image, you can paste relevant code here:

```
%%writefile train_lora.py
import os
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model
import torch
import numpy as np

# ===== Configuration - edit if needed =====
MODEL_NAME = "gpt2-medium" # switched from distilgpt2 -> gpt2-medium
OUTPUT_DIR = "lora-output"
TRAIN_FILE = "train_v2.jsonl"
VALID_FILE = "valid_v2.jsonl"
BATCH_SIZE = 1 # lower to fit gpt2-medium on 8GB
EPOCHS = 8 # increased from 3 -> 6
MAX_LENGTH = 512 # reduce context to save memory
GRADIENT_ACCUM_STEPS = 8 # accumulate grads to simulate larger batch
LEARNING_RATE = 2e-4
# ==========================================

if not (os.path.exists(TRAIN_FILE) and os.path.exists(VALID_FILE)):
    raise FileNotFoundError("Please upload train.jsonl and valid.jsonl into Colab working dir.")

# Load dataset
dataset = load_dataset("json", data_files={"train": TRAIN_FILE, "validation": VALID_FILE})

# Tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})

# Preprocess: encode prompt+completion together as input; labels = input_ids with pad -> -100
def preprocess_batch(batch):
    texts = [p + c for p, c in zip(batch["prompt"], batch["completion"])]
    enc = tokenizer(texts, truncation=True, padding="max_length", max_length=MAX_LENGTH)
    input_ids = enc["input_ids"]
    attention_mask = enc["attention_mask"]
    # labels: copy of input_ids, but replace pad_token_id with -100 so loss ignores them
    pad_id = tokenizer.pad_token_id
    labels = []
    for ids in input_ids:
        lab = [(i if i != pad_id else -100) for i in ids]
        labels.append(lab)
    enc["labels"] = labels
    return enc

tokenized = dataset.map(preprocess_batch, batched=True, remove_columns=dataset["train"].
    column_names)

# Data collator: simple collator (already padded)
def collate_fn(batch):
    # batch is list of dicts with input_ids, attention_mask, labels
    input_ids = torch.tensor([b["input_ids"] for b in batch], dtype=torch.long)
    attention_mask = torch.tensor([b["attention_mask"] for b in batch], dtype=torch.long)
    labels = torch.tensor([b["labels"] for b in batch], dtype=torch.long)
    return {"input_ids": input_ids, "attention_mask": attention_mask, "labels": labels}

device = "cuda" if torch.cuda.is_available() else "cpu"
```

```python
# Load base model
print(f"Loading base model {MODEL_NAME} ...")
model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
# If tokenizer length changed (we added pad token), resize token embeddings
if model.get_input_embeddings().weight.size(0) != len(tokenizer):
    print("Resizing token embeddings from", model.get_input_embeddings().weight.size(0), "to",
        len(tokenizer))
    model.resize_token_embeddings(len(tokenizer))

model = model.to(device)

# Attach LoRA (parameter-efficient)
lora_config = LoraConfig(r=8, lora_alpha=32, lora_dropout=0.1, bias="none", task_type="
    CAUSAL_LM")
model = get_peft_model(model, lora_config)

# Training arguments (minimal, with disabled reporting)
training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    gradient_accumulation_steps=GRADIENT_ACCUM_STEPS,
    num_train_epochs=EPOCHS,
    logging_steps=50,
    learning_rate=LEARNING_RATE,
    fp16=True,
    save_total_limit=2,
    report_to=["none"],
    eval_steps=200, # run evaluation every 200 steps
    save_steps=200, # save checkpoint every 200 steps
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized["train"],
    eval_dataset=tokenized["validation"],
    data_collator=collate_fn,
)

# Train
trainer.train()

# Evaluate explicitly
print("Running evaluation on validation set...")
metrics = trainer.evaluate(eval_dataset=tokenized["validation"])
print("Evaluation metrics:", metrics)

# Save adapter + tokenizer
model.save_pretrained(OUTPUT_DIR)
tokenizer.save_pretrained(OUTPUT_DIR)
print(" Training complete. Model and adapter saved to", OUTPUT_DIR)
```

8 Appendix: Example post process + UI snippet

```
# Single self-contained Gradio UI cell (paste & run in Colab)
!pip install -q gradio

import re, torch, gradio as gr
from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel

# ---------- Config ----------
MODEL_NAME = "gpt2-medium"
ADAPTER_DIR = "lora-output" # where your adapter is saved
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
# ----------------------------

# Load tokenizer + base model + adapter (resize embeddings if needed)
print("Loading tokenizer and model (this may take ~30-90s)...")
tokenizer = AutoTokenizer.from_pretrained(ADAPTER_DIR)
base = AutoModelForCausalLM.from_pretrained(MODEL_NAME)

if base.get_input_embeddings().weight.size(0) != len(tokenizer):
    print("Resizing embeddings:", base.get_input_embeddings().weight.size(0), "->", len(
        tokenizer))
    base.resize_token_embeddings(len(tokenizer))

# Load LoRA adapter
model = PeftModel.from_pretrained(base, ADAPTER_DIR).to(DEVICE)
model.eval()

# ---- Adapter sanity check ----
if hasattr(model, "peft_config"):
    print(" LoRA adapter loaded with config:", model.peft_config)
else:
    print(" Warning: No LoRA adapter detected, running base model only!")

print("Model loaded on", DEVICE)

# ---------- Safe generation helpers ----------
CLOSINGS = ["Sincerely", "Best regards", "Kind regards", "Regards", "With gratitude", "Thank
    you"]

def generate_email_safe(prompt,
                        max_new_tokens=150,
                        temperature=0.6,
                        top_p=0.9,
                        use_beam=False,
                        num_beams=4,
                        bad_words_ids=None):
    inputs = tokenizer(prompt, return_tensors="pt").to(DEVICE)
    gen_kwargs = dict(
        input_ids=inputs["input_ids"],
        attention_mask=inputs.get("attention_mask", None),
        max_new_tokens=int(max_new_tokens),
        pad_token_id=tokenizer.pad_token_id,
        eos_token_id=tokenizer.eos_token_id,
        no_repeat_ngram_size=3,
        repetition_penalty=1.4,
    )
    if bad_words_ids is not None:
        gen_kwargs["bad_words_ids"] = bad_words_ids

    try:
        if use_beam:
```

```
            out = model.generate(do_sample=False, num_beams=int(num_beams),
                                 early_stopping=True, **gen_kwargs)
        else:
            out = model.generate(do_sample=True, temperature=float(temperature),
                                 top_p=float(top_p), **gen_kwargs)
    except Exception as e:
        # fallback to safer short sampling if beam fails
        print(" Generation failed, falling back:", e)
        out = model.generate(input_ids=inputs["input_ids"], max_new_tokens=80,
                             do_sample=True, top_p=0.9, temperature=0.7,
                             pad_token_id=tokenizer.pad_token_id)
    raw = tokenizer.decode(out[0][inputs["input_ids"].shape[-1]:], skip_special_tokens=True)
    return raw.strip()

# (rest of cleanup_and_validate, generate_with_fallback, and Gradio UI stays same)


# ---------- Safe generation helpers ----------
CLOSINGS = ["Sincerely", "Best regards", "Kind regards", "Regards", "With gratitude", "Thank
    you"]

def generate_email_safe(prompt,
                        max_new_tokens=150,
                        temperature=0.6,
                        top_p=0.9,
                        use_beam=False,
                        num_beams=4,
                        bad_words_ids=None):
    inputs = tokenizer(prompt, return_tensors="pt").to(DEVICE)
    gen_kwargs = dict(
        input_ids=inputs["input_ids"],
        attention_mask=inputs.get("attention_mask", None),
        max_new_tokens=int(max_new_tokens),
        pad_token_id=tokenizer.pad_token_id,
        eos_token_id=tokenizer.eos_token_id,
        no_repeat_ngram_size=3,
        repetition_penalty=1.4,
    )
    if bad_words_ids is not None:
        gen_kwargs["bad_words_ids"] = bad_words_ids

    try:
        if use_beam:
            out = model.generate(do_sample=False, num_beams=int(num_beams),
                                 early_stopping=True, **gen_kwargs)
        else:
            out = model.generate(do_sample=True, temperature=float(temperature),
                                 top_p=float(top_p), **gen_kwargs)
    except Exception as e:
        # fallback to safer short sampling if beam fails
        out = model.generate(input_ids=inputs["input_ids"], max_new_tokens=80,
                             do_sample=True, top_p=0.9, temperature=0.7,
                             pad_token_id=tokenizer.pad_token_id)
    raw = tokenizer.decode(out[0][inputs["input_ids"].shape[-1]:], skip_special_tokens=True)
    return raw.strip()

def cleanup_and_validate(prompt, text):
    # Normalize and strip weird unicode
    text = text.strip()
    text = re.sub(r"\s+\n", "\n", text)
    text = re.sub(r"[^\x00-\x7F]+", " ", text)
    text = re.sub(r"\s{2,}", " ", text)
    text = text.replace("Subject :", "Subject:")
```

```
    # Ensure Subject
    if "Subject" not in text:
        m_sub = re.search(r"(extension|recommend|leave|clarification|thank|submit)", prompt,
             flags=re.I)
        subj = (m_sub.group(0).title() if m_sub else "Request")
        text = f"Subject: {subj}\n\n" + text

    # Ensure greeting uses professor name from prompt
    prof = None
    m = re.search(r"(Dr\.|Prof\.|Professor)\s+[A-Z][a-zA-Z]+", prompt)
    if m:
        prof = m.group(0)
        if "Dear" not in text:
            text = f"Dear {prof},\n\n" + text
        else:
            text = re.sub(r"Dear\s+[^\n,]+", f"Dear {prof}", text)

    # Heuristic: force wrong years -> 2025
    text = re.sub(r"\b(19|20)\d{2}\b", lambda x: ("2025" if x.group(0) != "2025" else "2025"),
         text)

    # Trim after polite closing
    for stop in CLOSINGS:
        if stop in text:
            text = text.split(stop)[0] + stop
            break

    text = text.strip().rstrip(".,'\"-")

    # Basic validation
    valid = True
    if prof is None:
        valid = False
    if "Subject" not in text:
        valid = False
    if not any(stop in text for stop in CLOSINGS):
        valid = False

    return text, valid

def generate_with_fallback(prompt, use_beam=True, num_beams=4, max_new_tokens=150, temperature
     =0.6, top_p=0.9):
    # Build a quick bad_words list to block a few problematic tokens (optional)
    bad_words = []
    for s in ["", "", "", "2010", "2015", "2018", "2020"]:
        enc = tokenizer.encode(s, add_special_tokens=False)
        if len(enc) > 0:
            bad_words.append(enc)
    bad_words_ids = bad_words if bad_words else None

    raw = generate_email_safe(prompt, max_new_tokens=max_new_tokens, temperature=temperature,
                              top_p=top_p, use_beam=use_beam, num_beams=num_beams,
                              bad_words_ids=bad_words_ids)
    cleaned, ok = cleanup_and_validate(prompt, raw)
    if ok:
        return cleaned
    # Safe templated fallback
    prof_m = re.search(r"(Dr\.|Prof\.|Professor)\s+[A-Z][a-zA-Z]+", prompt)
    prof = prof_m.group(0) if prof_m else "[Professor]"
    date_m = re.search(r"\b(January|February|March|April|May|June|July|August|September|October|
         November|December)\s+\d{1,2},\s*\d{4}\b", prompt)
    date = date_m.group(0) if date_m else "[DATE]"
```

```
    if "extension" in prompt.lower():
        subject = f"Subject: Request for 3-Day Extension on {date}"
        body = f"Dear {prof},\n\nI hope you are well. I have been unwell and was unable to complete the assignment by the original deadline. I respectfully request a 3-day extension and will submit by {date}.\n\nSincerely,\n[Your Name]"
    elif "recommend" in prompt.lower():
        subject = "Subject: Request for Recommendation Letter"
        body = f"Dear {prof},\n\nI hope you are well. I am applying for an internship with deadline {date} and would be grateful if you could provide a letter of recommendation. I can share my resume if needed.\n\nBest regards,\n[Your Name]"
    else:
        subject = "Subject: Request"
        body = f"Dear {prof},\n\n{prompt}\n\n{date}\n\nSincerely,\n[Your Name]"
    return subject + "\n\n" + body

# ---------- Gradio UI ----------
examples = [
    "Instruction: Draft an email to Dr. Nair asking for a 3-day extension on the Cybersecurity assignment due to illness. Include subject line, greeting, reason, new submission date October 10, 2025, and polite closing.",
    "Instruction: Write a polite email to Prof. Das requesting a letter of recommendation for an internship. Include subject line, greeting, purpose, deadline September 18, 2025, and polite closing.",
    "Instruction: Write a formal email to Dr. Verma requesting leave of absence until October 5, 2025 due to family emergency. Include subject and polite closing."
]

def ui_generate(prompt, temperature, max_tokens, use_beam, beams):
    return generate_with_fallback(prompt, use_beam=use_beam, num_beams=beams,
                                  max_new_tokens=max_tokens, temperature=temperature)

with gr.Blocks() as demo:
    gr.Markdown("##  Email Drafter (LoRA)  production-safe")
    with gr.Row():
        inp = gr.Textbox(lines=6, label="Instruction / Prompt", placeholder=examples[0])
        with gr.Column(scale=0.4):
            temp = gr.Slider(0.2, 1.0, value=0.6, step=0.05, label="Temperature")
            max_tokens = gr.Slider(50, 400, value=200, step=10, label="Max new tokens")
            beam_switch = gr.Checkbox(label="Use beam search (deterministic)", value=True)
            beams = gr.Slider(1, 8, value=4, step=1, label="Num beams")
            gen_btn = gr.Button("Generate")
    out = gr.Textbox(label="Generated Email", lines=12)

    gen_btn.click(fn=ui_generate, inputs=[inp, temp, max_tokens, beam_switch, beams], outputs=out)

demo.launch(share=True)
```