Here’s a high-level recap of our entire journey so far:

1. **Project Goal & Initial Plan**  
   – You’re building a web app where users query a restaurant’s health-inspection violations, then get consumer-friendly summaries and keywords generated by a transformer.  
   – To make that work, you needed a fine-tuned model that maps real “Violations” text → structured summaries + keywords.
2. **First Pass: CSV Splits & Free-form Labels**  
   – Sampled 3,000 “Violations” rows from the Chicago health-inspection CSV.  
   – Split into train/val/test (80/10/10) and saved as CSV.  
   – Wrote a process\_violation function that calls the OpenAI chat API to generate a single blob (summary + keywords) for each violation.  
   – Ran those in parallel, captured the LLM output in one Excel column, then loaded it back for fine-tuning.
3. **Improving Label Structure: JSONL & Error Handling**  
   – Switched the API prompt to return a **strict JSON object** with exactly two keys (summary, keywords).  
   – Changed our pipeline to drop any "[ERROR]" failures and write out **JSONL** files (violations\_train.jsonl, violations\_val.jsonl, violations\_test.jsonl), one JSON object per line.  
   – This gave us clean, machine-readable data that fits directly into Hugging Face’s load\_dataset("json", …) workflows.
4. **Analyzing & Balancing Labels**  
   – Ran a quick counter on the summary verdicts (“safe”, “may unsafe”, etc.) and saw a heavy skew toward “may make it unsafe” and only one strict “not safe” example.  
   – Decided to augment the rare “not safe to eat” class so the model can learn it.
5. **Synthetic Augmentation Pipeline**  
   – **generate\_synthetic\_violations(n\_samples)**: Used the API to invent brand-new inspector-style comments that clearly warrant “not safe to eat here.”  
   – **enrich\_synthetic\_violations(...)**: Fed those synthetic texts back into the JSON-prompt pipeline so they get summaries + keywords with the forced “not safe” verdict.  
   – **merge\_jsonl\_files(...)**: Concatenated the original violations\_train.jsonl + violations\_synthetic.jsonl into a shuffled violations\_train\_final.jsonl—our rock-solid, augmented training set.  
   – Left violations\_val.jsonl and violations\_test.jsonl untouched so evaluation remains honest.
6. **Next Step: Fine-Tuning**  
   – We now have three ready-to-use JSONL splits:  
   • violations\_train\_final.jsonl (real + synthetic)  
   • violations\_val.jsonl  
   • violations\_test.jsonl  
   – The plan is to adapt your professor’s “Fine-Tuning LLMs for Domain-Specific Use” notebook:
   1. Load those JSONL files via datasets.load\_dataset("json", …)
   2. Map each example into the model’s input/output format (e.g. sequence-to-sequence or JSON-string targets)
   3. Apply PEFT (LoRA/QLoRA) training arguments
   4. Evaluate with ROUGE, BERTScore, and qualitative checks