LLM Course Project

Medical Question Generation via Fine-Tuning

Training a Language Model to Ask the Right Questions

February 12, 2026

Contents

1. Project Goal

The goal of this project is to train a language model that **asks the right questions**. Given a partial patient summary (a few sentences describing a patient's clinical situation), the model outputs *one* focused, clinically relevant question—specifically the question whose answer would most help identify the causative pathogen responsible for a suspected bloodstream infection.

2. Architecture Overview

The pipeline involves four models, labelled **A**, **B**, **C**, and **D**, across three broad phases: dataset generation, dataset labelling, and model training/evaluation.

2.1. Phase 1 — Dataset Generation (Models A & B)

Two large, already-pretrained LLMs (e.g. `gpt-4o`) serve as model A and model B.

1. **Model A** receives the full raw clinical data for a patient and is instructed to produce a *partial* medical summary in which only ~50% of the most important information is exposed.
2. **Model B** receives the partial summary from A and is instructed to ask *one single, focused* clinical question about the most important piece of missing information.
3. **Model A** then answers B's question using the complete raw data (so the answer is accurate and detailed).

Each such exchange produces three artefacts:

- x — the initial partial medical summary.
- q — the question that was asked.
- $d = x + q +$ answer — the full dialogue as a single concatenated string.

2.2. Phase 2 — Labelling Good Questions (Model C)

A third pretrained LLM, **model C**, acts as a pathogen classifier. For each case it is asked to identify the causative bacterium given some text input. It is evaluated twice:

1. On the partial summary alone (x).
2. On the full dialogue (d).

The accuracy metric is defined as the rank of the ground-truth pathogen in C's ordered list of predictions. If C achieves *higher accuracy* on d than on x, the question q is considered **"good"** and the pair $[x, q]$ is saved.

2.3. Phase 3 — Fine-Tuning & Evaluation (Model D)

A small, already-pretrained LLM (e.g. `Mistral-7B`) is **fine-tuned** on the collected $[x, q]$ pairs via *auto-regressive* training, so that it learns to generate an appropriate question given a partial medical summary.

For evaluation, model D receives unseen summaries x from the test set, produces a predicted question q^*, and the similarity between q^* and the ground-truth q is measured with several metrics.

3. Data Extraction

The raw data comes from the **MIMIC-IV** dataset. We extracted all cases corresponding to bloodstream infections, yielding approximately **14,000 cases**.

```
Output path: /app/code/LLM_project/bsi-agent/data/processed/bsi_cases.jsonl
Max cases: 14000
--------------------------------------------------
Loading microbiology events...
Loading patients...
Loading admissions...
/app/code/LLM_project/bsi-agent/src/bsi_agent/data/bsi_cohort.py:197: FutureWarning: DataFrameG
s deprecated, and in a future version of pandas the grouping columns will be excluded from the
roupings or explicitly select the grouping columns after groupby to silence this warning.
  has_suscept = positive_bc.groupby("micro_specimen_id").apply(
Found 14751 positive blood cultures
Processing 14000 BSI cases...
Loading lab events...
Loading prescriptions...
Loading vital signs (this may take a while)...
Extracted 14000 complete BSI cases
Saved 14000 cases to /app/code/LLM_project/bsi-agent/data/processed/bsi_cases.jsonl
--------------------------------------------------
Extracted 14000 BSI cases

Top 10 pathogens:
  STAPH AUREUS COAG +: 2940
  ESCHERICHIA COLI: 2448
  STAPHYLOCOCCUS EPIDERMIDIS: 1528
  STAPHYLOCOCCUS, COAGULASE NEGATIVE: 1319
  ENTEROCOCCUS FAECIUM: 928
  KLEBSIELLA PNEUMONIAE: 891
  ENTEROCOCCUS FAECALIS: 548
  PSEUDOMONAS AERUGINOSA: 324
  BETA STREPTOCOCCUS GROUP B: 224
  STREPTOCOCCUS PNEUMONIAE: 173
```

Figure 1: Distribution of MIMIC-IV bloodstream infection cases used as the raw data source.

4. Validating Model C on Full Information

Before generating the training dataset, we first verified that model C (`gpt-4.1-nano`) could classify the causative pathogen reliably when given *all* clinical information. We generated full medical summaries for 100 cases using the following prompt:

```
SUMMARY_PROMPT = """You are a clinical documentation specialist. Given the following structured patient data,
 write a comprehensive medical summary for a patient with a suspected bloodstream infection.

IMPORTANT RULES:
- ONLY describe the information that IS present in the data below.
- Do NOT mention, note, or comment on any categories of information that are absent, missing, or unavailable. Simply
 omit them from your summary entirely.
- Do NOT write phrases like "no medications are specified", "gram stain was not available", "vitals are not reported", etc.
- Write as if the provided data is the complete clinical picture available at this time.
- Do NOT mention the actual pathogen/organism if shown - that is for diagnosis.

Style Instruction:
{style}

Patient Data:
{patient_data}

Write a detailed clinical summary using ONLY the information provided above and following the style instruction:"""
```

Figure 2: Prompt used to generate full medical summaries from raw patient data. The same prompt is reused for partial summaries (with hidden data).

4.1. Style Augmentation

To ensure diversity in the generated summaries, we implemented a `StylesGenerator` class that randomly samples a writing style (e.g. formal, casual, concise) and appends it to each summarisation request.

```python
def generate_styles(self, temperature: float = 0.3) -> dict:

    response = self.client.chat.completions.create(
        model=self.model,
        messages=[{"role": "user", "content": STYLES_PROMPT}],
        temperature=temperature,
        max_tokens=1500,
        response_format={"type": "json_object"}
    )

    print("STYLES FOR SUMMARIES:")
    print(response.choices[0].message.content)

    json_output = response.choices[0].message.content.strip()

    self.styles = json.loads(json_output)

    return self.styles
```

Figure 3: `StylesGenerator` class holding various writing styles. Calling the `get_style()` method triggers a GPT request to generate the style attributes.

```
STYLES_PROMPT = """
Create attributes describing how a synthetic medical summary for
a patient with suspected bloodstream infection may vary.

Requirements:
- Focus only on writing style, tone, and content structure.
- Do NOT include medical facts or clinical decisions.
- Provide exactly 5 attributes.
- Each attribute should contain 3–5 distinct possible values.

Return output strictly as valid JSON in the following format:
{
    "attribute_name": ["value1", "value2", "value3"]
}
"""
```

Figure 4: GPT prompt used internally by `StylesGenerator` to obtain style attributes for a given style name.

```
STYLES FOR SUMMARIES:
{
    "writing_style": ["formal", "informal", "technical", "narrative"],
    "tone": ["neutral", "empathetic", "urgent", "reassuring"],
    "content_structure": ["chronological", "problem-solution", "cause-effect", "priority-based"],
    "level_of_detail": ["highly detailed", "moderately detailed", "brief", "bullet points"],
    "use_of_jargon": ["extensive", "minimal", "balanced", "none"]
}
```

Figure 5: Example style attributes returned by GPT (e.g. tone, vocabulary level, sentence structure).

```
def sample_random_style_string(self) -> str:
    """
    Randomly select one value from each style attribute
    and return a formatted string.
    """

    if not self.styles:
        raise ValueError("Styles were not generated yet. Call generate_styles() first.")

    selections = []

    for key, values in self.styles.items():
        chosen_value = random.choice(values)
        selections.append(f"{key}: {chosen_value}")

    return ", ".join(selections)
```

Figure 6: Function that randomly samples one style from the available options.

```python
styles_gen = StylesGenerator(api_key=api_key, model=model)
styles = styles_gen.generate_styles()

# Initialize generator
generator = SummaryGenerator(api_key=api_key, model=model)

# Generate summaries
summaries = []
for case in tqdm(cases, desc="Generating summaries"):
    try:
        summary = generator.generate_summary(case, styles_gen.sample_random_style_string())
        summary = sanitize_summary(summary, case.get("organism", ""))
        summaries.append({
            "case_id": case["case_id"],
            "ground_truth_pathogen": case["organism"],
            "full_summary": summary,
        })
    except Exception as e:
        print(f"\nError processing {case['case_id']}: {e}")
        continue
```

Figure 7: Integration of the randomly sampled style into the medical summary generation request.

4.2. Results on 100 Cases

Running the code on 100 cases produced the following output:

```
root@a7808db3a190:/app/code/LLM_project/bsi-agent# python scripts/generate_summaries.py --max_cases 100
Loading cases from: /app/code/LLM_project/bsi-agent/data/processed/bsi_cases.jsonl
Processing 100 cases with gpt-4.1-nano
--------------------------------------------------
STYLES FOR SUMMARIES:
{
    "writing_style": ["formal and technical", "concise and straightforward", "detailed and elaborative", "neutral and objective", "simplified and layman-frie
ndly"],
    "tone": ["professional and serious", "empathetic and reassuring", "detached and clinical", "urgent and alerting", "calm and reassuring"],
    "content_structure": ["chronological sequence", "sectioned with headings", "problem-solution format", "summary-focused", "question-and-answer style"],
    "use_of_jargon": ["extensive medical terminology", "minimal technical language", "balanced with lay explanations", "abundant abbreviations", "plain langu
age with definitions"],
    "level_of_detail": ["highly detailed with comprehensive descriptions", "moderately detailed highlighting key points", "brief overview with essential infc
rmation", "focused on specific sections only", "general summary with minimal specifics"]
}
Generating summaries:  93%|
Generating summaries:  94%|
Generating summaries:  95%|
Generating summaries: 100%|| 100/100 [09:34<00:00,  5.75s/it]
--------------------------------------------------
Saved 100 summaries to /app/code/LLM_project/bsi-agent/data/processed/full_summaries.jsonl
```

Figure 8: Console output after generating full medical summaries for 100 cases.

An example of a generated full medical summary:

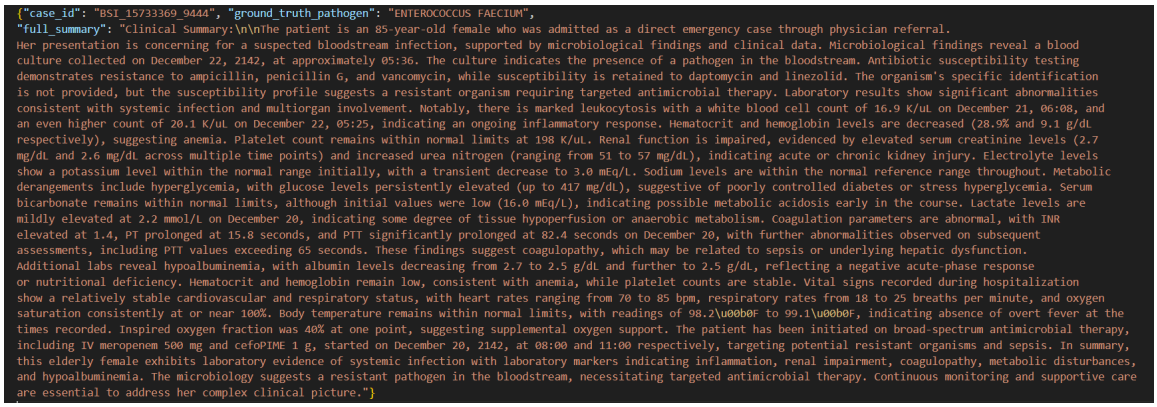

Figure 9: Sample full medical summary generated by the model.

We then evaluated model C's classification accuracy:

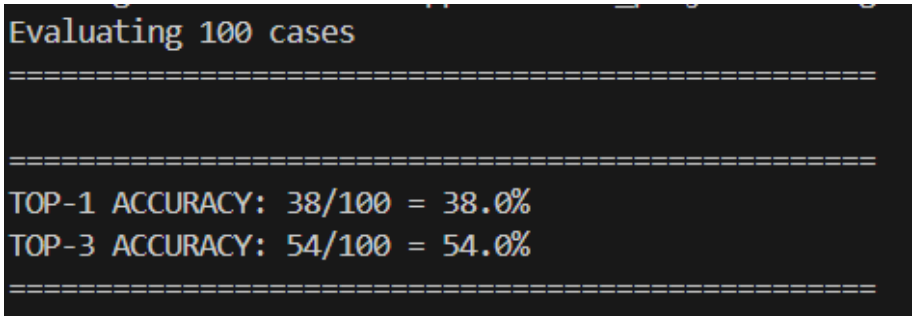

Figure 10: Top-1 and Top-3 accuracy of model C when given full summaries. Top-1: 39%, Top-3: 55%.

```
------------------------------------------------
Breakdown by pathogen:
  [OK] ESCHERICHIA COLI: top1=22/29 (76%)  top3=28/29 (97%)
  [LOW] STAPH AUREUS COAG +: top1=11/22 (50%)  top3=14/22 (64%)
  [LOW] STAPHYLOCOCCUS, COAGULASE NEGATIVE: top1=0/9 (0%)  top3=0/9 (0%)
  [OK] KLEBSIELLA PNEUMONIAE: top1=1/7 (14%)  top3=7/7 (100%)
  [LOW] STAPHYLOCOCCUS EPIDERMIDIS: top1=0/7 (0%)  top3=2/7 (29%)
  [LOW] ENTEROCOCCUS FAECIUM: top1=3/5 (60%)  top3=3/5 (60%)
  [OK] ENTEROCOCCUS FAECALIS: top1=0/4 (0%)  top3=4/4 (100%)
  [OK] SERRATIA MARCESCENS: top1=0/3 (0%)  top3=3/3 (100%)
  [LOW] STAPHYLOCOCCUS HOMINIS: top1=0/2 (0%)  top3=0/2 (0%)
  [LOW] KLEBSIELLA OXYTOCA: top1=0/2 (0%)  top3=0/2 (0%)
  [LOW] BETA STREPTOCOCCUS GROUP B: top1=0/1 (0%)  top3=0/1 (0%)
  [LOW] STREPTOCOCCUS ANGINOSUS (MILLERI) GROUP: top1=0/1 (0%)  top3=0/1 (0%)
  [LOW] AEROCOCCUS VIRIDANS: top1=0/1 (0%)  top3=0/1 (0%)
  [LOW] MORGANELLA MORGANII: top1=0/1 (0%)  top3=0/1 (0%)
  [LOW] STAPHYLOCOCCUS HAEMOLYTICUS: top1=0/1 (0%)  top3=0/1 (0%)
  [OK] PROTEUS MIRABILIS: top1=0/1 (0%)  top3=1/1 (100%)
  [LOW] STREPTOCOCCUS SANGUINIS: top1=0/1 (0%)  top3=0/1 (0%)
  [OK] STENOTROPHOMONAS MALTOPHILIA: top1=0/1 (0%)  top3=1/1 (100%)
  [OK] ENTEROBACTER CLOACAE COMPLEX: top1=0/1 (0%)  top3=1/1 (100%)
  [OK] PSEUDOMONAS AERUGINOSA: top1=1/1 (100%)  top3=1/1 (100%)
```

Figure 11: Detailed classification results per pathogen category.

Takeaway: Top-1 accuracy of 39% and Top-3 accuracy of 55% on 100 cases. The team noted that switching to a stronger classifier model might be worthwhile.

5. Dataset Creation

5.1. Step 1 — Generating Partial Medical Summaries

For every case we hide a random fraction of the raw clinical data before generating the summary. The hiding distribution is shown below:

```python
# Probability of KEEPING a specific item within a category.
# 1.0 = Always keep
# 0.0 = Always remove
# 0.4 = Keep roughly 40% of the items (lines) in that list
ITEM_KEEP_PROBABILITY = {
    # Atomic Fields (Single values)
    "demographics": 1.0,      # ALWAYS KEEP: Age/Gender are fundamental context
    "admission": 1.0,         # ALWAYS KEEP: Context (ICU vs Ward) is vital

    # List Fields (We will filter these lists)
    "labs": 0.4,              # Keep only 40% of labs -> Forces model to ask for missing ones
    "vitals": 0.5,            # Keep 50% of vitals
    "medications": 0.5,       # Keep 50% of meds

    # Sensitive Fields (Must be hidden in Partial Summary to prevent easy guessing)
    "gram_stain": 0.0,        # ALWAYS HIDE in partial
    "susceptibilities": 0.0,  # ALWAYS HIDE in partial (Leak prevention)
    "organism": 0.0           # ALWAYS HIDE in partial (The Answer)
}
```

Figure 12: Probability distribution used to decide which data fields are hidden for each case. For example, `gram_stain` is *always* hidden because it directly implies the pathogen identity; other fields are hidden with varying probabilities.

After hiding, the partial data is fed into model A together with a style instruction (same prompt as Figure 2) to produce the partial summary x.

5.2. Step 2 — Generating Questions

Each partial summary x is then passed to model B, which is instructed to ask exactly one focused clinical question. The prompt used is:

```
QUESTION_PROMPT = """You are an infectious disease specialist reviewing a partial patient summary.

Writing Instructions:
{style_instructions}

Current Status:
The patient summary below is incomplete.

**AVAILABLE DATA HINTS:**
The following specific data points exist in the full record but are currently hidden from you. **You
 MUST ask about one of these topics** to ensure the record can provide an answer:
[ {available_hints} ]

Your task is to ask ONE specific question about one of the missing items listed above that is most critical for diagnosis.

Partial Summary:
{partial_summary}

Ask a single, specific clinical question targeting the available hidden info:
"""
```

Figure 13: Prompt used to generate a single clinical question from a partial medical summary.

Style augmentation is also applied to question generation, with a dedicated style prompt to match question-writing conventions:

```
QUESTION_STYLES_PROMPT = """
Create attributes describing how a diagnostic clinical question
(asked by an infectious disease specialist about a suspected bloodstream infection) may vary.

Requirements:
- Focus only on how the QUESTION is phrased.
- Do NOT include medical facts or clinical decisions.
- Do NOT include specific pathogens or diagnoses.
- Provide exactly 5 attributes.
- Each attribute must describe a dimension of question style, tone, or reasoning approach.
- Each attribute should contain 3–5 distinct possible values.

Examples of relevant variation dimensions include:
- Question structure
- Specificity level
- Clinical reasoning focus
- Urgency or tone
- Information targeting strategy

Return output strictly as valid JSON in the following format:
{
    "attribute_name": ["value1", "value2", "value3"]
}
"""
```

Figure 14: Style prompt tailored for generating diverse question styles (e.g. direct, elaborate, terse).

```
STYLES FOR QUESTIONS:
{
  "question_structure": ["open-ended", "closed-ended", "multi-part"],
  "specificity_level": ["broad/general", "focused/specific", "highly detailed"],
  "clinical_reasoning_focus": ["etiology", "diagnostic process", "management implications"],
  "urgency_or_tone": ["urgent/urgent-sounding", "neutral", "cautious/considerate"],
  "information_targeting_strategy": ["broad patient context", "specific clinical findings", "laboratory data emphasis"]
}
```

Figure 15: Example question style attributes.

Note on data leakage: We also provide the model with explicit hints about which fields are *entirely absent* from the raw MIMIC-IV data for a given case. Without these hints, the model frequently asked about fields that did not exist at all, wasting the question allocation. This does *not* constitute a data leak because our goal is purely to generate high-quality $[x, q]$ pairs for training a smaller model—the hints are not used at inference time.

5.3. Step 3 — Generating Answers

Model A is called again, this time in the role of a knowledgeable responder. It receives the *complete* raw clinical data, the question q, and an instruction to answer concisely. The prompt:

```
ANSWER_PROMPT = """You are a clinical assistant with access to the complete patient record.
Your task is to answer the specific clinical question based **ONLY** on the provided data.

CRITICAL RULES:
1. **Objective Data Only**: Provide specific numbers, values, and observations (e.g., "WBC is 14.5", "Temperature 39.1°C").
2. **Handle Missing Data**: If the requested information is NOT in the patient record below, you must state:
 "This information is not available in the record." Do NOT guess or assume normal values.
3. **No Interpretations**: Do not say "suggestive of sepsis" or "consistent with infection." Just state the facts.
4. **Redaction Protocol**: Do NOT name the specific pathogen (bacteria/fungus) even if the data implies it.

Complete Patient Record:
{patient_data}

Question: {question}

Focused Answer (Objective facts only):"""
```

Figure 16: Prompt for answer generation. Model A sees the full raw data and must answer the specific question asked.

An example of a complete dialogue:


```
{"case_id": "BSI_14066157_8784", "ground_truth_pathogen": "STAPHYLOCOCCUS, COAGULASE NEGATIVE",

 "x": "This 67-year-old male patient was admitted emergently through the emergency room at 03:57 on February 28, 2126, with clinical suspicion of a
 bloodstream infection. Blood cultures were obtained on March 2, 2126, at 07:48. Laboratory data reveal significant abnormalities indicative of systemic
 infection and multiorgan involvement. Notably, the patient exhibits marked leukocytosis with white blood cell counts escalating from 14.8 K/uL to over
 23 K/uL, confirmed by replicate analysis, supporting an ongoing infectious process. Hematocrit levels are decreased initially at 30.6%, further declining
 to 32.7% and subsequently to 8.8 g/dL hemoglobin, indicating anemia, likely secondary to systemic illness or ongoing blood loss. Renal function is impaired,
 with serum creatinine rising from 1.6 mg/dL to 1.4 mg/dL, and urea nitrogen elevated substantially at 43.0 and 47.0 mg/dL, suggestive of acute kidney injury
or prerenal azotemia. Electrolyte levels remain relatively stable, with sodium, potassium, and chloride within normal ranges, although bicarbonate is slightly
decreased at 21.0 mEq/L, indicating a possible metabolic acidosis component. Lactate levels are persistently elevated, initially at 3.8 mmol/L and increasing
to 4.5 mmol/L, then decreasing to 1.9 mmol/L, reflecting episodes of tissue hypoperfusion or sepsis-related metabolic derangement. Coagulation parameters are
abnormal, with INR elevated to 1.6 initially, progressing to 1.5 and 1.3, and PTT markedly prolonged up to 80.9 seconds, indicating coagulopathy. Blood glucose
is elevated at 195.0 mg/dL initially, with subsequent fluctuations, and hemoglobin remains low throughout, consistent with anemia of critical illness. Vital
signs recorded during the initial assessment show variable blood pressures, with systolic readings ranging from 96 to 123 mmHg and diastolic pressures from 46
to 54 mmHg, alongside a temperature of approximately 100.8\u00b0F, suggestive of febrile response. Heart rate varies from 70 to 84 bpm, and respiratory rate
fluctuates between 7 and 20 breaths per minute, with pulse oximetry saturation consistently high at 99-100%. Inspired oxygen fraction was maintained at 40%,
indicating supplemental oxygen support. Therapeutically, the patient received intravenous vancomycin at 1000 mg starting at 08:00 on February 28, 2126, as part
of empiric antimicrobial coverage. The clinical picture, characterized by leukocytosis, elevated lactate, renal impairment, coagulopathy, anemia, and persistent
fever, underscores a severe systemic inflammatory response consistent with a suspected bloodstream infection. The blood culture obtained is critical for pathogen
identification and guiding targeted therapy. Continuous monitoring of hemodynamic and laboratory parameters remains essential to assess progression and response to treatment.",

"q": "What is the patient's current arterial blood pressure (systolic, diastolic, and mean) and heart rate, as these are critical for assessing hemodynamic stability
in the context of suspected sepsis?",

"answer": "Arterial blood pressure (systolic): 96.0 mmHg  \nArterial blood pressure (diastolic): 51.0 mmHg  \nArterial blood pressure (mean): 71.0 mmHg
\nHeart rate: 78.0 bpm",


"d": "This 67-year-old male patient was admitted emergently through the emergency room at 03:57 on February 28, 2126, with clinical suspicion of a bloodstream infection.
Blood cultures were obtained on March 2, 2126, at 07:48. Laboratory data reveal significant abnormalities indicative of systemic infection and multiorgan involvement.
Notably, the patient exhibits marked leukocytosis with white blood cell counts escalating from 14.8 K/uL to over 23 K/uL, confirmed by replicate analysis, supporting an
ongoing infectious process. Hematocrit levels are decreased initially at 30.6%, further declining to 32.7% and subsequently to 8.8 g/dL hemoglobin, indicating anemia,
likely secondary to systemic illness or ongoing blood loss. Renal function is impaired, with serum creatinine rising from 1.6 mg/dL to 1.4 mg/dL, and urea nitrogen elevated
substantially at 43.0 and 47.0 mg/dL, suggestive of acute kidney injury or prerenal azotemia. Electrolyte levels remain relatively stable, with sodium, potassium, and chloride
```


Figure 17: Example full dialogue: partial summary (x), question (q), and answer, forming the complete dialogue string d.

5.4. Step 4 — Selecting Good Questions

Model C is called twice for each case — once on x and once on d — using the classification prompt shown below:

```
CLASSIFIER_PROMPT = """You are an infectious disease specialist. Based on the following clinical summary
of a patient with a bloodstream infection, predict the most likely causative pathogen(s).

Clinical Summary:
{summary}

Provide your prediction as a ranked list of the top 10 most likely pathogens:
1. [Pathogen name]
2. [Pathogen name]
3. [Pathogen name]
4. [Pathogen name]
5. [Pathogen name]
6. [Pathogen name]
7. [Pathogen name]
8. [Pathogen name]
9. [Pathogen name]
10. [Pathogen name]

Use standard microbiological nomenclature (e.g., "Staphylococcus aureus", "Escherichia coli", "Klebsiella pneumoniae").
Only output the numbered list, nothing else."""
```

Figure 18: Classification prompt used to evaluate whether a question improves pathogen identification.

The model's response is parsed into an ordered prediction list:


```
{"case_id": "BSI_19489119_13119", "ground_truth": "ENTEROCOCCUS FAECALIS",
"predictions_x": ["Escherichia coli", "Klebsiella pneumoniae", "Pseudomonas aeruginosa", "Enterococcus faecalis",
"Staphylococcus aureus", "Acinetobacter baumannii", "Candida albicans", "Streptococcus pneumoniae", "Enterobacter cloacae",
 "Serratia marcescens"], "rank_x": 4}

{"case_id": "BSI_19489119_13119", "ground_truth": "ENTEROCOCCUS FAECALIS",
"predictions_d": ["Escherichia coli", "Klebsiella pneumoniae", "Pseudomonas aeruginosa", "Enterococcus faecalis", "Staphylococcus aureus",
 "Streptococcus pneumoniae", "Acinetobacter baumannii", "Enterobacter cloacae", "Candida albicans", "Listeria monocytogenes"], "rank_d": 4}
```


Figure 19: Example parsed prediction list. The *rank* is the position of the ground-truth pathogen in this list.

A question is saved as *good* if $\text{rank}(d) > \text{rank}(x)$, i.e. the dialogue provided more useful information to the classifier than the summary alone.

5.5. Step 5 — Running at Scale

The entire pipeline was run over all **14,000 cases**. The results are shown below:

Figure 20: Pipeline run statistics — case counts and processing details.

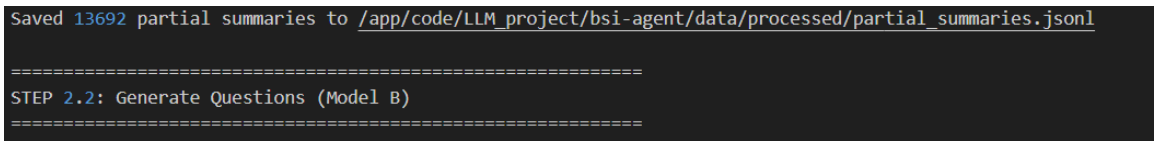

Figure 21: Summary statistics after filtering for good questions.

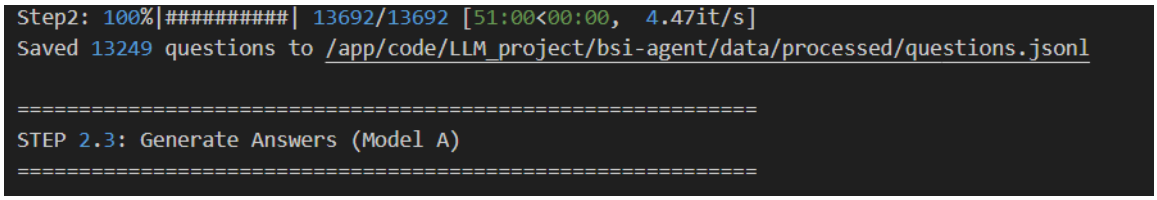

Figure 22: Breakdown of question quality across the full dataset.

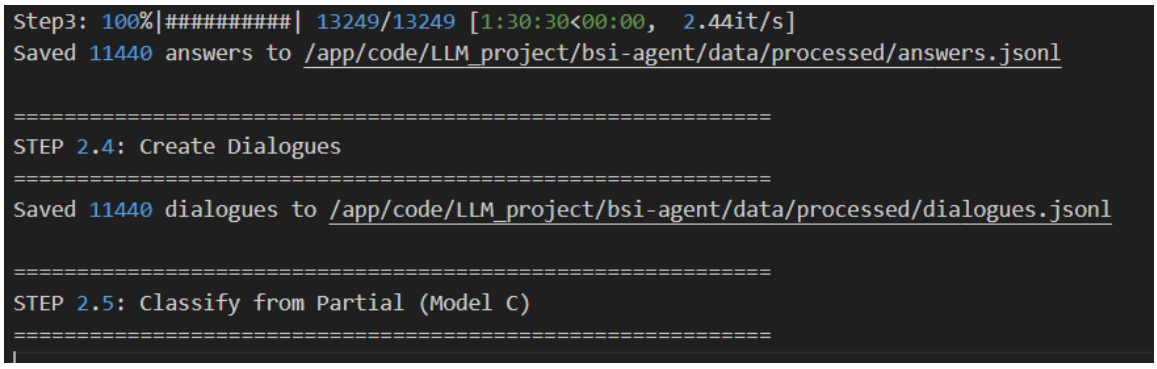

Figure 23: Pathogen distribution among good questions.

```
Saved 11240 classifications to /app/code/LLM_project/bsi-agent/data/processed/classifications_x.jsonl
Accuracy from x: 3781/11240 = 33.6%

============================================================
STEP 2.6: Classify from Dialogue (Model C)
============================================================
```

Figure 24: Additional quality metrics from the pipeline run.

```
Saved 11204 classifications to /app/code/LLM_project/bsi-agent/data/processed/classifications_d.jsonl
Accuracy from d: 3593/11204 = 32.1%

============================================================
STEP 2.7: Filter Good Questions
============================================================
Saved 1106 good questions to /app/code/LLM_project/bsi-agent/data/processed/good_questions.jsonl

Pipeline complete.
```

Figure 25: Final count: **1,106 good questions** collected and saved as $[x, q]$ pairs in a JSONL file.

Cost: The entire data generation pipeline used `gpt-4.1-nano` and cost approximately $8.

6. Model Training

6.1. Dataset Preparation

The $[x, q]$ pairs are loaded from the JSONL file. A task-specific *instruction header* is prepended to each summary x to form the model input:

```python
x = item.get("x") or item.get("partial_summary", "")
q = item.get("q") or item.get("question", "")

if not x or not q:
    return None

user_content = QUESTION_PROMPT_TRAINING.format(partial_summary=x)

messages = [
    {"role": "user", "content": user_content},
    {"role": "assistant", "content": q},
]

prompt_only = [
    {"role": "user", "content": user_content},
]
```

Figure 26: Code that loads the pairs and prepends the instruction header to each partial summary.

```
QUESTION_PROMPT_TRAINING = """You are an infectious disease specialist reviewing a partial patient summary
 for a patient with a suspected bloodstream infection.

The summary below has specific missing data points (such as specific labs, vitals, or history) that are critical for diagnosis.

Your task is to ask ONE specific clinical question about a missing value that would help identify the causative pathogen.

Focus on asking about:
- Specific missing Laboratory values (e.g., Lactate, WBC, Platelets, Creatinine)
- Missing Vital signs (e.g., Temperature, Blood Pressure)
- Specific risk factors (e.g., indwelling lines, recent surgery, immunosuppression)
- The source of infection
- Culture results or Gram stain
- Antibiotic susceptibility or resistance profiles (AST)

IMPORTANT - Do NOT ask about:
- The name of the organism or pathogen
- General/vague questions like "How is the patient?"

Partial Summary:
{partial_summary}

Ask a single, specific clinical question (just the question, no explanation):"""
```

Figure 27: The instruction header that tells the small model what it should do. This becomes the *input* to the model; the corresponding q becomes the *target output*.

The tokenizer is configured to assign a label of -100 to all instruction tokens so the model is *not* trained to predict them (only the question tokens contribute to the loss):

```python
# Tokenize full conversation (user + assistant response)
formatted = tokenizer.apply_chat_template(
    messages,
    tokenize=True,
    add_generation_prompt=False,
    max_length=max_length,
    truncation=True,
    return_dict=True,
)

# Tokenize prompt only (with generation prompt = assistant header)
# to find where the assistant response starts
prompt_formatted = tokenizer.apply_chat_template(
    prompt_only,
    tokenize=True,
    add_generation_prompt=True,
    return_dict=True,
)

input_ids = formatted["input_ids"]
attention_mask = formatted.get("attention_mask", [1] * len(input_ids))
prompt_len = len(prompt_formatted["input_ids"])

# Mask labels: -100 for prompt tokens, real ids for assistant response only
labels = [-100] * min(prompt_len, len(input_ids)) + input_ids[prompt_len:]
```

Figure 28: Tokenization code. Instruction tokens are masked with -100 so they are excluded from the auto-regressive loss.

A diagram of the full data preparation flow:

```python
# Load data
print("\nLoading data...")
data = load_jsonl(data_path)
print(f"Loaded {len(data)} items")

# Load tokenizer
print("\nLoading tokenizer...")
tokenizer = AutoTokenizer.from_pretrained(base_model, trust_remote_code=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

# Prepare dataset
print("\nPreparing dataset...")
if args.mode == "xq":
    dataset = prepare_xq_dataset(data, tokenizer, max_length=max_seq_length)
else:
    dataset = prepare_dialogue_dataset(data, tokenizer, max_length=max_seq_length)

if len(dataset) == 0:
    print("Error: No valid training examples after processing")
    return

# Split into train/val
if len(dataset) > 1:
    split = dataset.train_test_split(test_size=max(1, int(len(dataset) * 0.1)), seed=42)
    train_dataset = split["train"]
    val_dataset = split["test"]
else:
    train_dataset = dataset
    val_dataset = dataset
print(f"Train: {len(train_dataset)}, Val: {len(val_dataset)}")
```

Figure 29: End-to-end data preparation pipeline before fine-tuning.

6.2. Model Loading and Quantisation

We load **Mistral-7B** from HuggingFace with 4-bit quantisation (QLoRA) to enable efficient fine-tuning:

```python
# Configure quantization
if not args.no_quantize and torch.cuda.is_available():
    print("\nConfiguring quantization (4-bit)...")
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_use_double_quant=True,
    )
else:
    bnb_config = None

# Load model
print(f"\nLoading model: {base_model}...")
model_kwargs = {
    "trust_remote_code": True,
    "torch_dtype": torch.float16 if torch.cuda.is_available() else torch.float32,
}
if bnb_config:
    model_kwargs["quantization_config"] = bnb_config
    model_kwargs["device_map"] = "auto"
else:
    model_kwargs["device_map"] = "auto" if torch.cuda.is_available() else None

model = AutoModelForCausalLM.from_pretrained(base_model, **model_kwargs)

if bnb_config:
    model = prepare_model_for_kbit_training(model)
else:
    # Enable gradients for non-quantized training
    model.enable_input_require_grads()
```

Figure 30: Model loading code with 4-bit quantisation configuration.

6.3. LoRA Adapter

We attach a **LoRA** adapter (`rank=16`) to the model:

```python
# Configure LoRA
print("\nConfiguring LoRA...")
lora_config = LoraConfig(
    r=lora_r,
    lora_alpha=lora_config_dict.get("lora_alpha", 32),
    lora_dropout=lora_config_dict.get("lora_dropout", 0.05),
    target_modules=lora_config_dict.get("target_modules", ["q_proj", "k_proj", "v_proj", "o_proj"]),
    bias="none",
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

Figure 31: LoRA configuration with rank 16.

6.4. Training

```python
# Trainer
print("\nInitializing trainer...")
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator,
)
```

Figure 32: Training arguments and `Trainer` initialisation.

Training was performed on an **NVIDIA L40 GPU**:

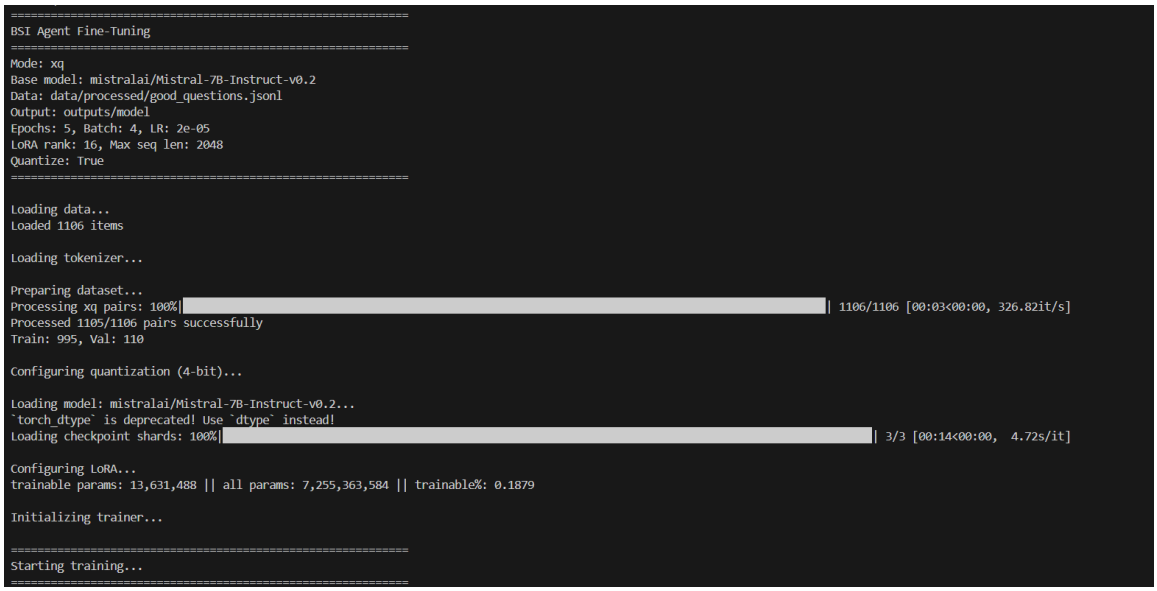

Figure 33: Training logs showing loss curves on the NVIDIA L40 GPU.

The resulting trained LoRA adapter:

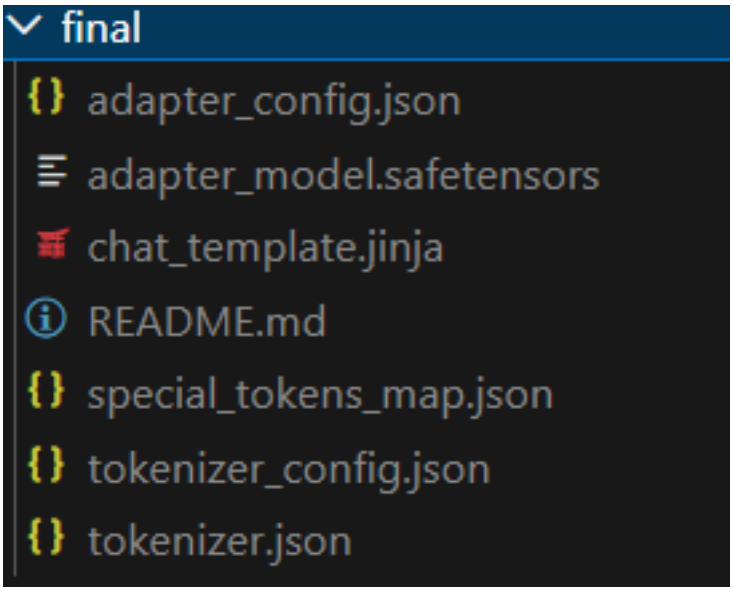

Figure 34: Files comprising the saved LoRA adapter after training.

7. Model Evaluation

After training on 90% of the good-question dataset, the model is evaluated on the remaining 10% (110 pairs). For each pair, model D receives x (with the instruction header) and generates a predicted question q^*.

An example $[q, q^*]$ pair:

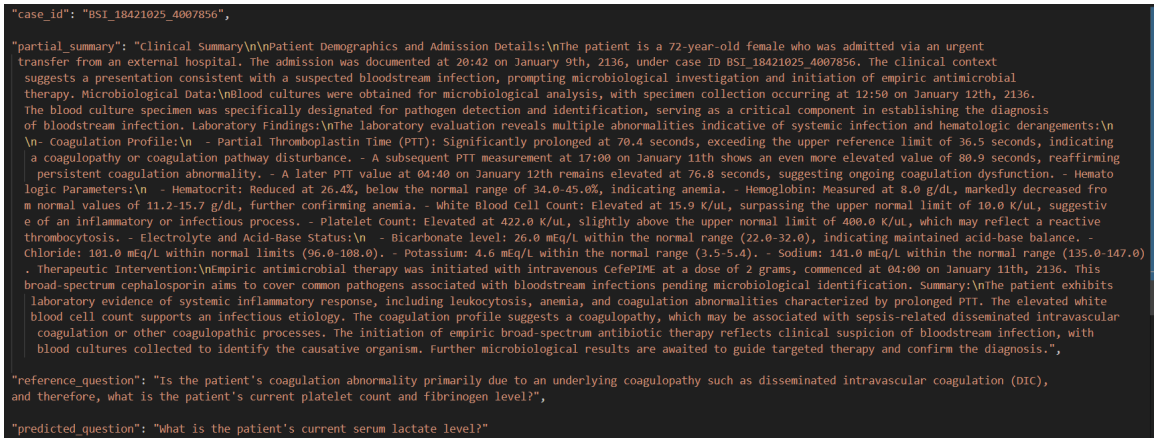

Figure 35: Example ground-truth question q vs. model-generated question q^* on a held-out test case.

The similarity between q^* and q is measured with four metrics:

- **Sentence Similarity** — cosine similarity between sentence embeddings (e.g. using a Sentence-BERT model).
- **BLEU** — n-gram precision of the generated question relative to the reference.
- **ROUGE-L** — longest common subsequence recall/precision.

- **BERTScore F1** — token-level semantic similarity using contextual BERT embeddings.

Results:

```
Metrics:
  sentence_similarity: 0.6076499223709106
  bleu: 0.2266185834490647
  rouge_l: 0.43255030899948793
  bertscore_f1: 0.9018909335136414
```

Figure 36: Evaluation results across the four metrics on the 110 held-out test pairs.

8. Implementation Notes and Observations

8.1. Hiding Probabilities

Key design decisions made during development:

1. **Gram stain always hidden.** Because gram stain results strongly imply pathogen identity, they are always excluded from the partial summary. This forces the model to ask more informative questions.
2. **Question prompt guards.** Model B initially tended to ask directly about culture/-gram stain results — information that model A could not provide since it was already hidden. A guard clause was added to the question prompt in `question_generator.py` to forbid asking about gram stain morphology.
3. **Answer generator uses raw data.** The `answer_generator.py` was updated so that model A answers questions from the full raw data (not from the full summary), allowing it to answer a broader range of questions.
4. **Statistical hiding via categories.** The `generate_training_data.py` script applies hiding at the *category level*: a probability of 0.2 for `labs` means there is an 80% chance the entire labs section is dropped. An improvement under consideration is to pass only a random 20% of the lab values rather than dropping the entire category.
5. **Number of top predictions.** The classifier was updated to return 10 ranked predictions (`pathogen_classifier.py`).

8.2. Results at the 100-Case Scale

- Initial run: **14 good questions** out of 100 cases.
- After adding the gram stain guard and updating hiding probabilities: **19 good questions** out of 100 cases.

9. Next Steps

1. **Style diversity for evaluation.** Style augmentation has been added to training data generation but needs to be applied consistently during the evaluation phase as well.

2. **Missing-field hints.** Explicitly telling the model which fields are absent in the raw data (not just hidden) avoids questions that can never be answered.
3. **Scale to full dataset.** Run the pipeline over all 14,000 cases targeting ∼2,000 good questions.
4. **Local training.** Fine-tune model D locally on the collected pairs.
5. **Style augmentation (future).** After the core pipeline is complete, augment the dataset with additional writing styles (friendly, scattered, professional, etc.).
6. **Variable hiding percentages (future).** Repeat the process with different hiding ratios (20%, 50%, 70%) via a wrapper script.