Harnessing large language models to improve antimicrobial use monitoring in dairy farming

B. Koppe[1], X. Yang[1], K.J. Koebel[2],
D.V. Nydam[3], M. Capel[4], R. Ivanek[2], J.J. Sun[1]

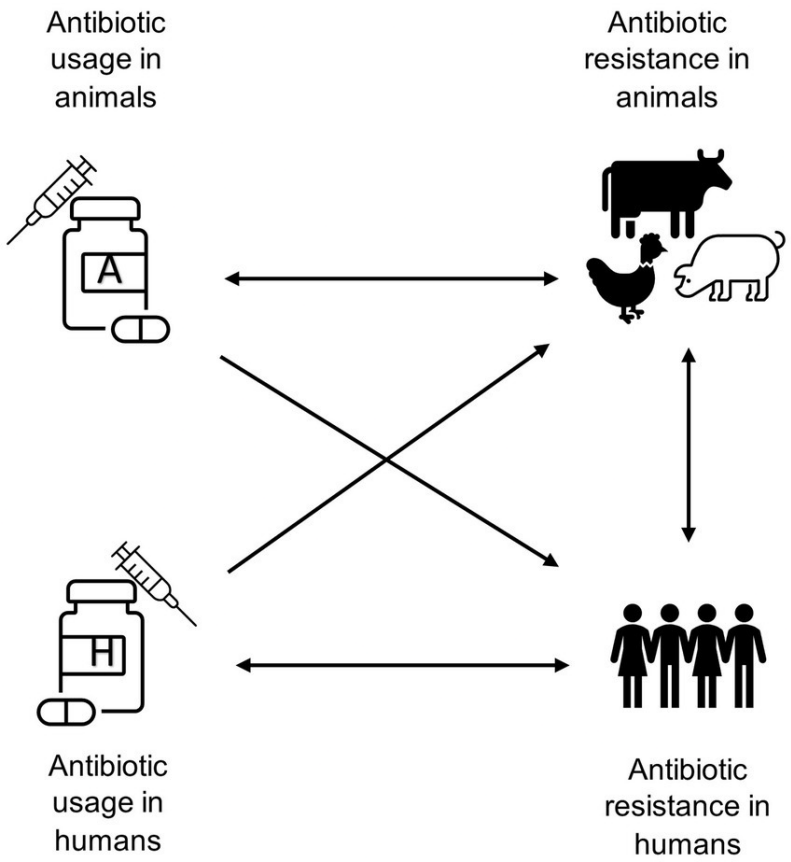
[1]Dept. Computer Science, Cornell Bowers College of Computing and Information Science, Ithaca NY

[2]Dept. Population Medicine & Diagnostic Sciences, Cornell University College of Veterinary Medicine, Ithaca NY

[3]Dept. Public & Ecosystem Health, Cornell University College of Veterinary Medicine

[4]Perry Veterinary Clinic, Perry NY

Background: Antimicrobials & Antibiotics



How can we better track & measure this data?

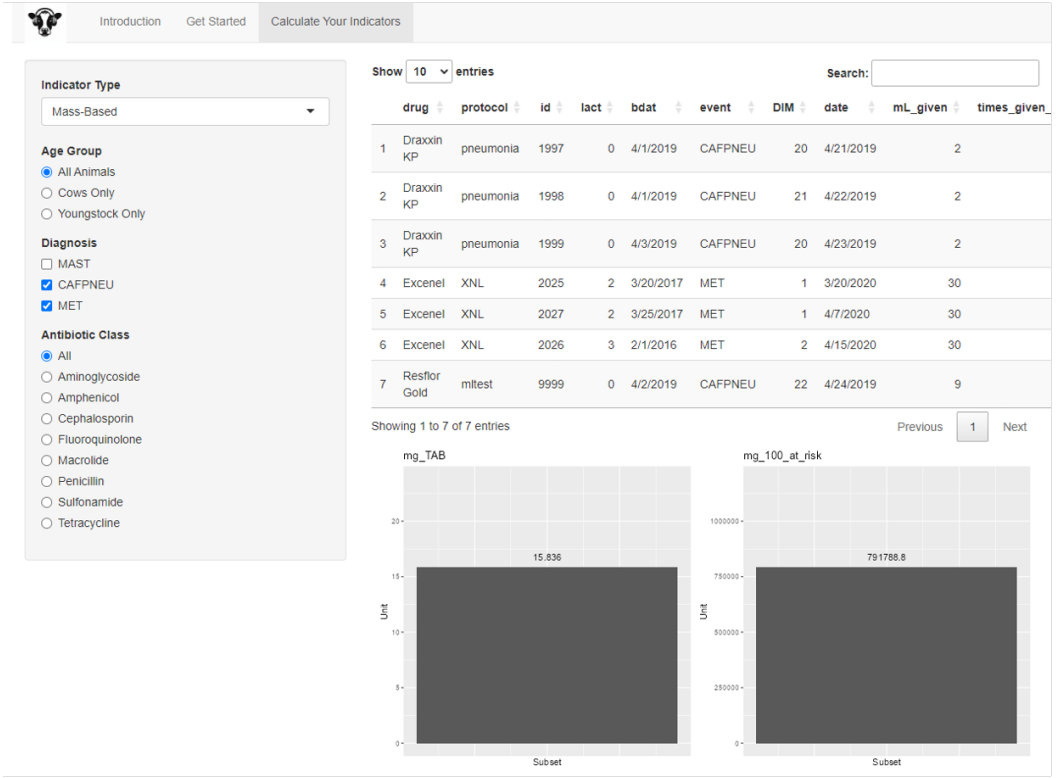
Background: AUDIO System

Dr. Katherine Koebel

Dr. Renata Ivanek

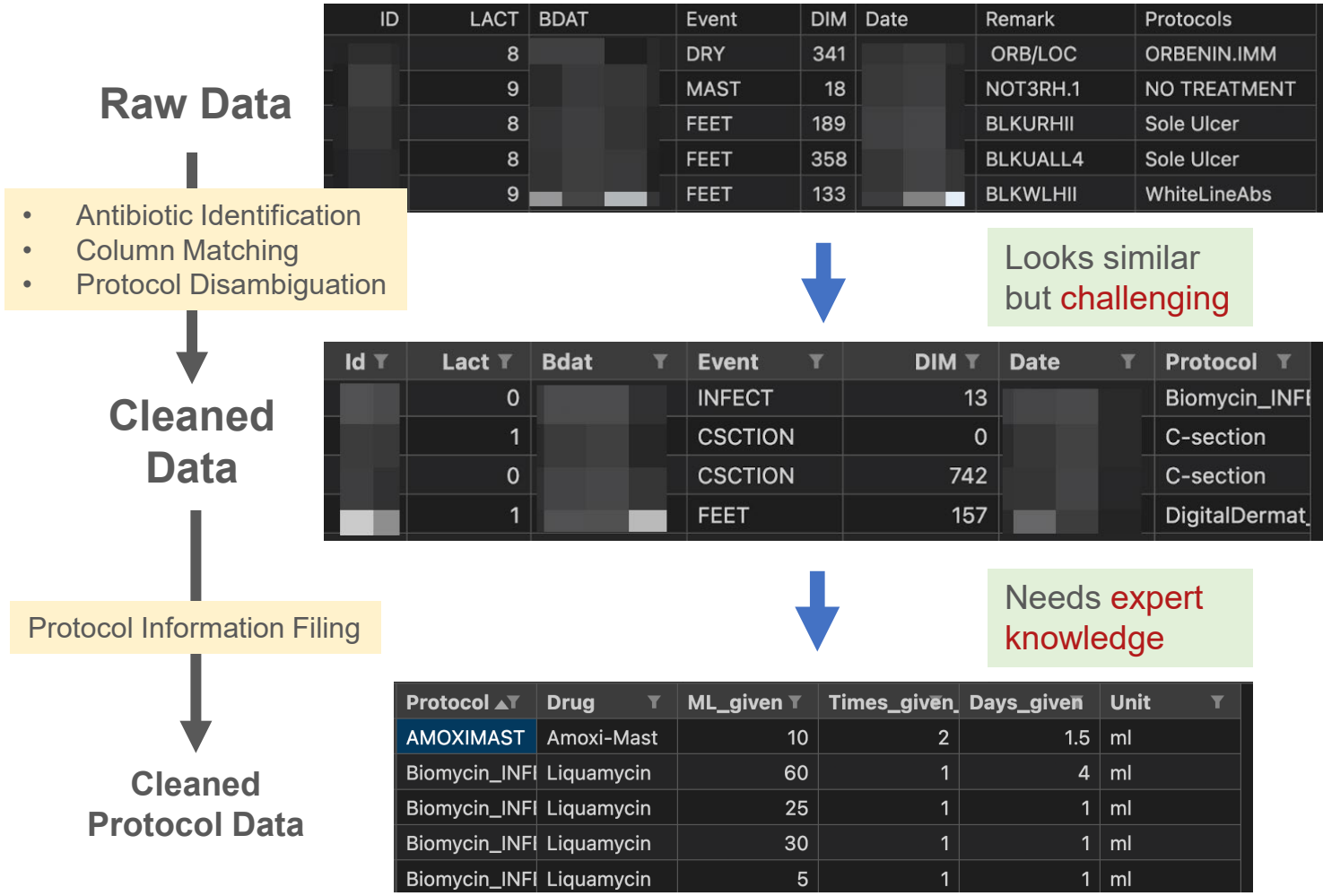
AUDIO

ANTIBIOTIC USE IN DAIRY INVESTIGATED ONLINE



AUDIO's Input Conversion

- Start with raw data corresponding to "cow events"
- Each row represents something that happened to a cow
  - foot trim, antibiotic administration, etc.
- We must find all rows that relate to antibiotic administration and extract antibiotic use
  - drug, amount administered, etc.
  - medically important antimicrobials



ID	LACT	BDAT	Event	DIM	Date	Remark	Protocols
	8		DRY	341		ORB/LOC	ORBENIN.IMM
	9		MAST	18		NOT3RH.1	NO TREATMENT
	8		FEET	189		BLKURHII	Sole Ulcer
	8		FEET	358		BLKUALL4	Sole Ulcer
	9		FEET	133		BLKWLHII	WhiteLineAbs

Id	Lact	Bdat	Event	DIM	Date	Protocol
	0		INFECT	13		Biomycin_INFI
	1		CSCTION	0		C-section
	0		CSCTION	742		C-section
	1		FEET	157		DigitalDermat

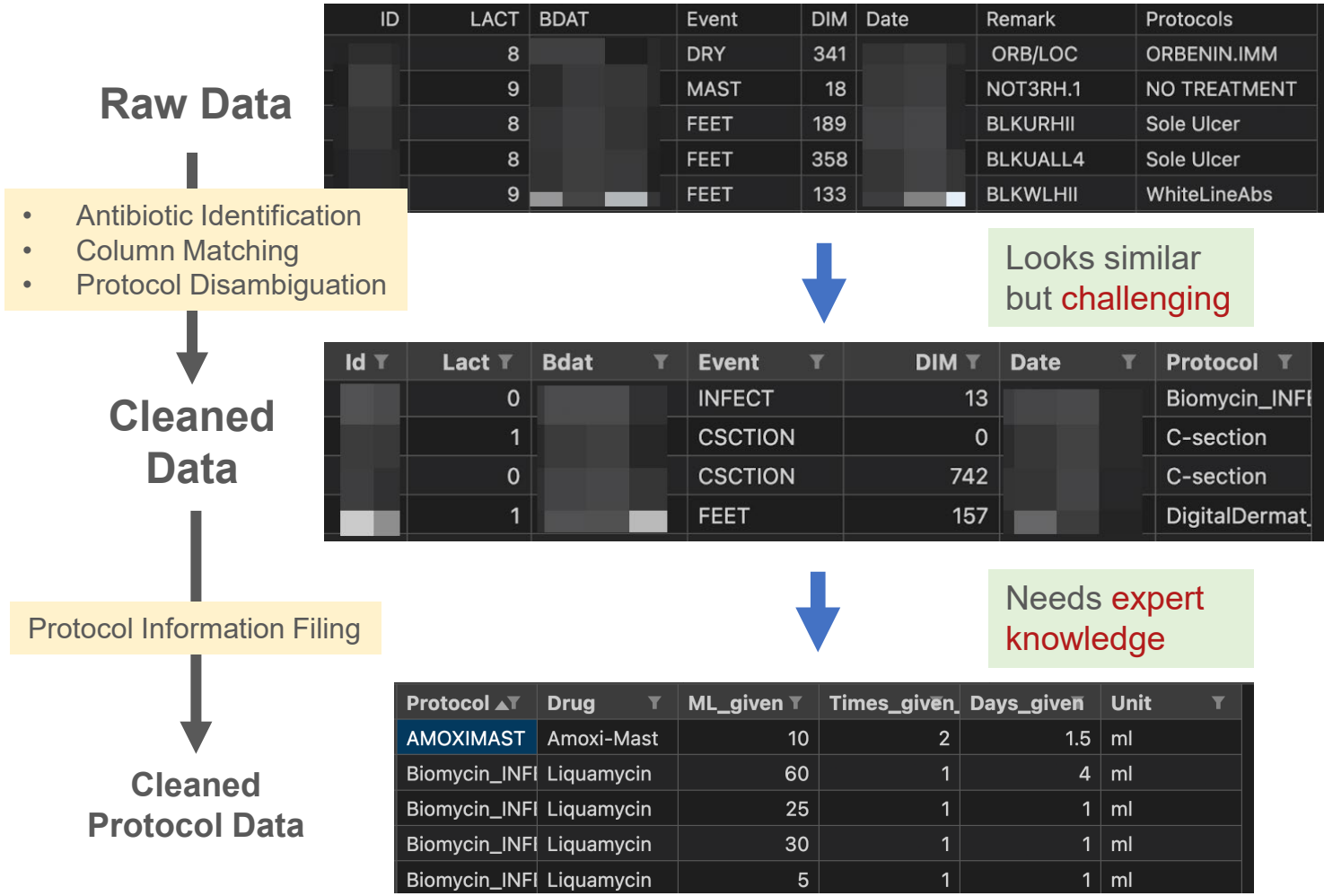
Protocol	Drug	ML_given	Times_given	Days_given	Unit
AMOXIMAST	Amoxi-Mast	10	2	1.5	ml
Biomycin_INFI	Liquamycin	60	1	4	ml
Biomycin_INFI	Liquamycin	25	1	1	ml
Biomycin_INFI	Liquamycin	30	1	1	ml
Biomycin_INFI	Liquamycin	5	1	1	ml

Key Challenges

- Sparse raw data
- Required field knowledge
- Drug information

Bottom line: Significant inference is required.

Lots of human effort needed.



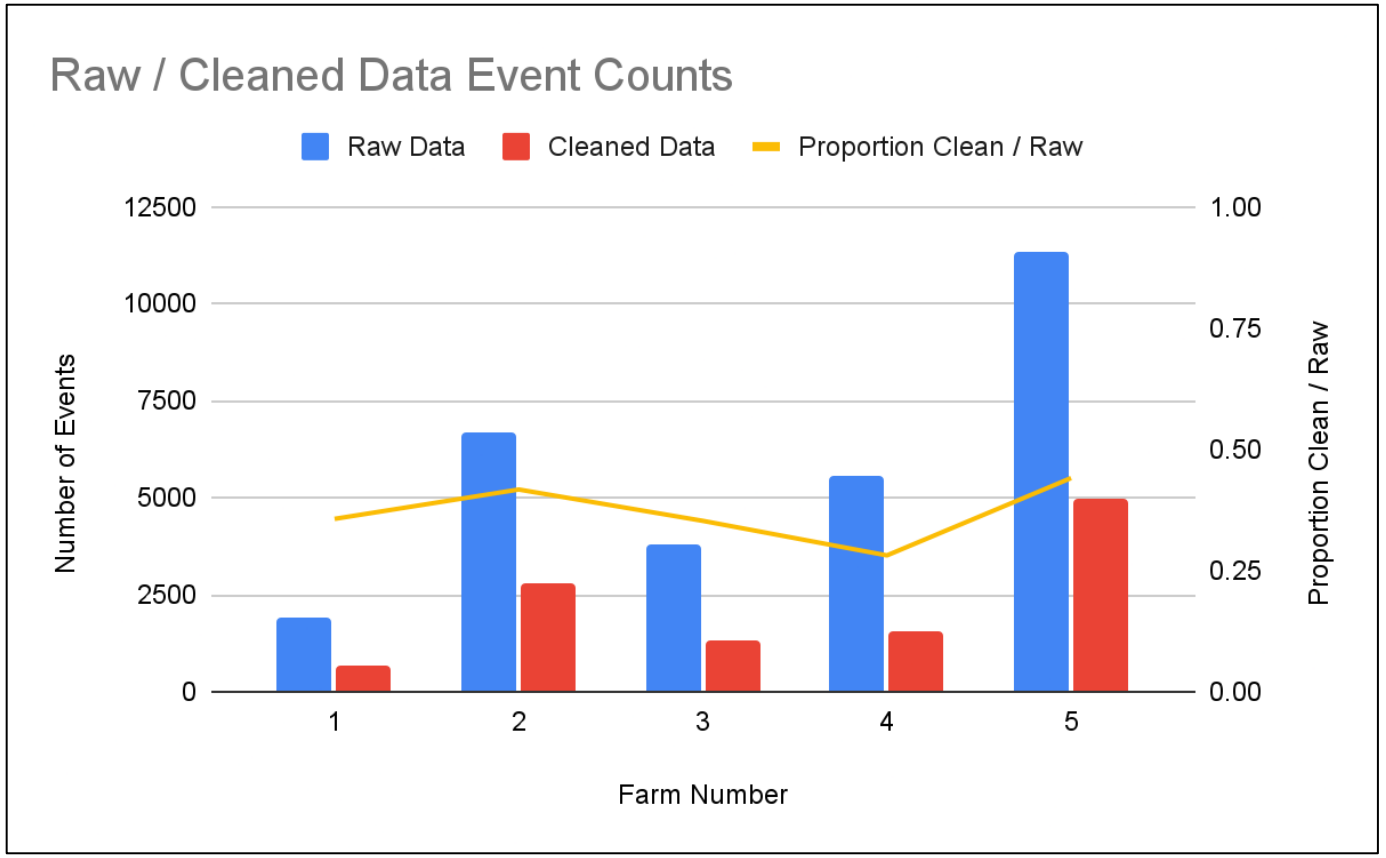
ID	LACT	BDAT	Event	DIM	Date	Remark	Protocols
	8		DRY	341		ORB/LOC	ORBENIN.IMM
	9		MAST	18		NOT3RH.1	NO TREATMENT
	8		FEET	189		BLKURHII	Sole Ulcer
	8		FEET	358		BLKUALL4	Sole Ulcer
	9		FEET	133		BLKWLHII	WhiteLineAbs

Id	Lact	Bdat	Event	DIM	Date	Protocol
	0		INFECT	13		Biomycin_INFI
	1		CSCTION	0		C-section
	0		CSCTION	742		C-section
	1		FEET	157		DigitalDermat

Protocol	Drug	ML_given	Times_given	Days_given	Unit
AMOXIMAST	Amoxi-Mast	10	2	1.5	ml
Biomycin_INFI	Liquamycin	60	1	4	ml
Biomycin_INFI	Liquamycin	25	1	1	ml
Biomycin_INFI	Liquamycin	30	1	1	ml
Biomycin_INFI	Liquamycin	5	1	1	ml

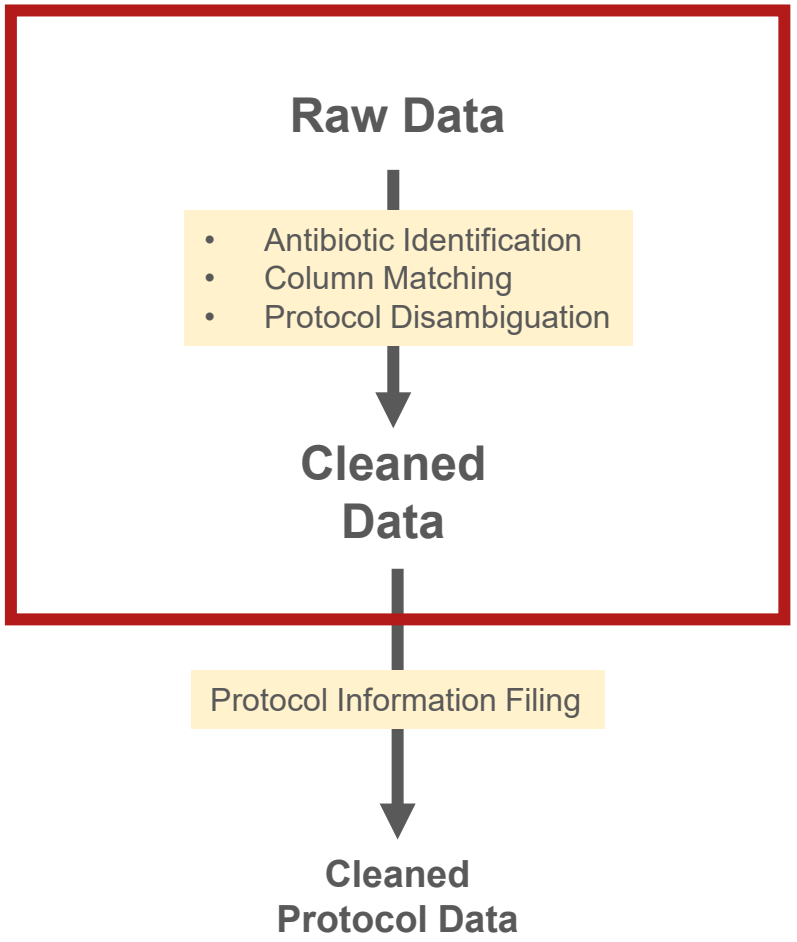
Five Farms

- Currently working with 1 year of data from N = 5 New York farms.
- Data size (event count) can differ significantly by farm
- We want to extract about the same proportion of rows from each farm.

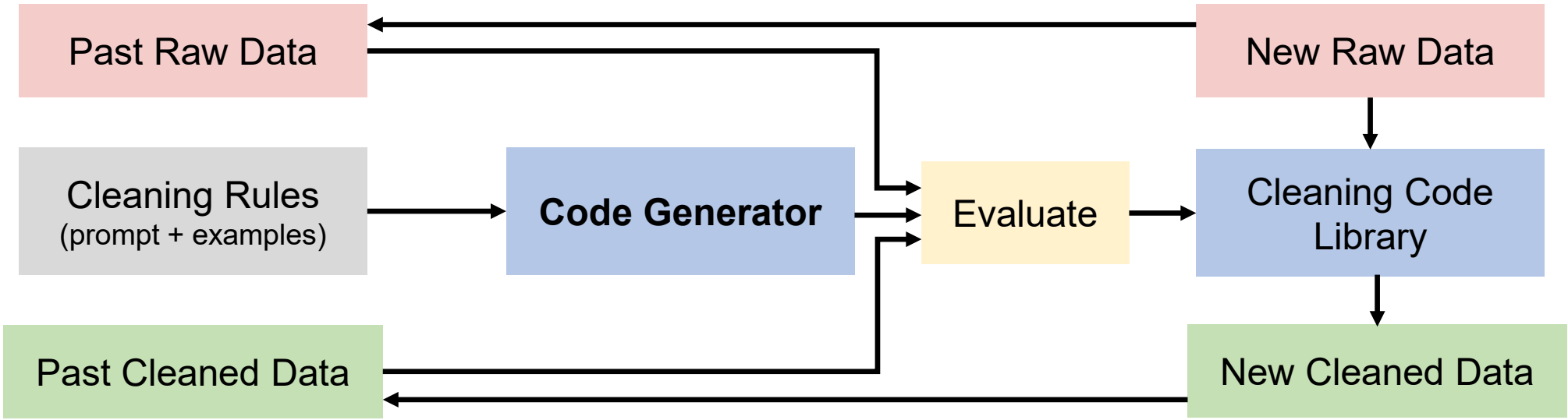


Methods

- Previous approach: Pure human labeling
  - Completely by hand
  - Tedious, time consuming
  - Cannot scale
- Goal: Leverage LLM for data standardization
  - Generalizable, scalable, saves time
  - Previous work can kickstart our LLM
  - Priority: usability for veterinary researchers



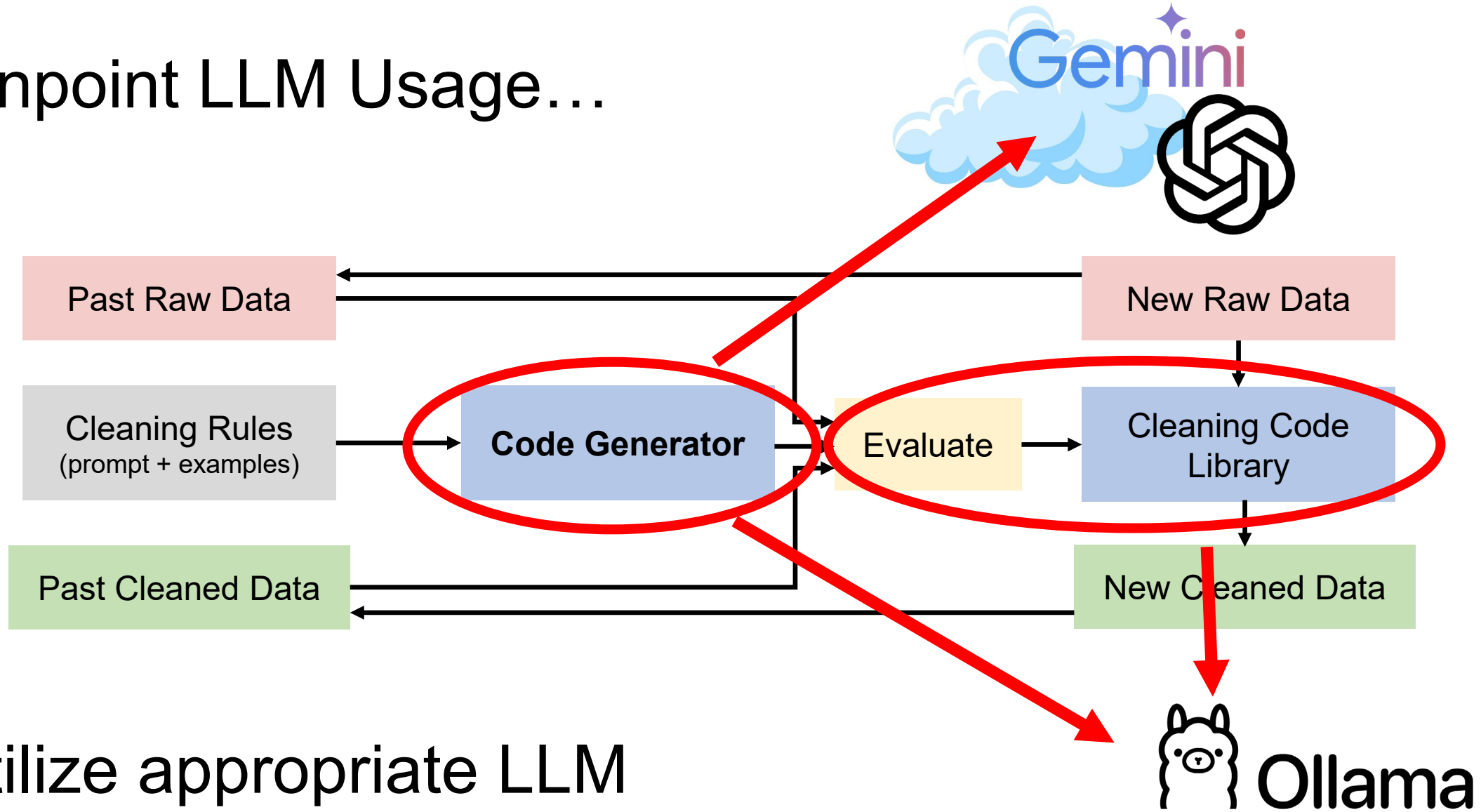
Agentic Pipeline



Hurdle: Confidential Data

- Farms' data is <u>confidential</u>
  - Cannot be given to mainstream AI providers (ChatGPT, Google Gemini, etc.)
- Solution: Host open-source models with Ollama
  - Meta Llama 3.3 70B
  - Ollama is easily portable, allowing us to easily swap from a weak to a powerful machine
  - Cloud providers still cannot be used

Pinpoint LLM Usage…

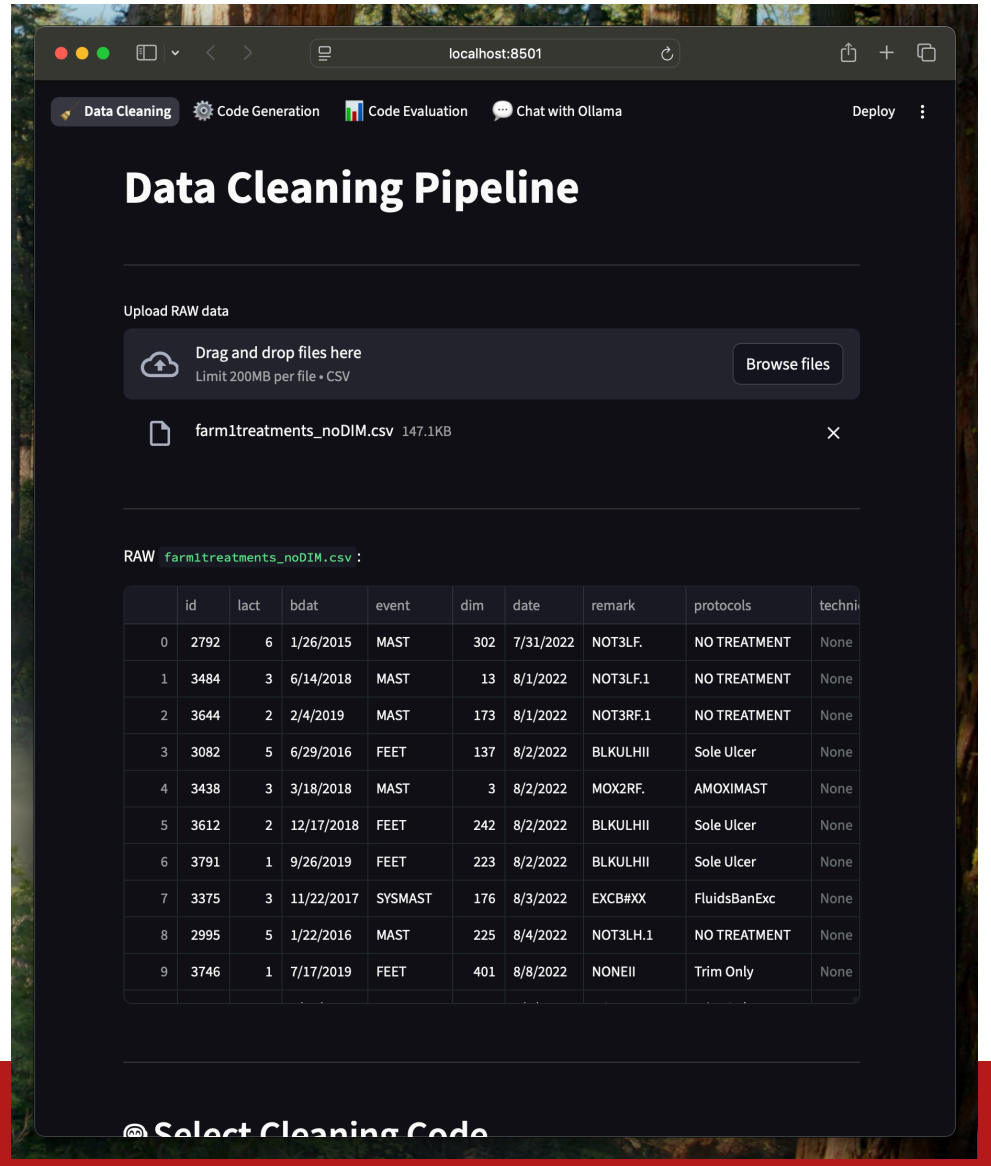


Utilize appropriate LLM

Hurdle: Usability

- Many moving parts
  - Generated files, multiple LLMs, etc.
- Intended to be used by veterinary researchers
  - Ideally could be used directly on-site, at a farm
- Solution: Website interface
  - Simple buttons and drag-and-drop interface

https://<URL>:



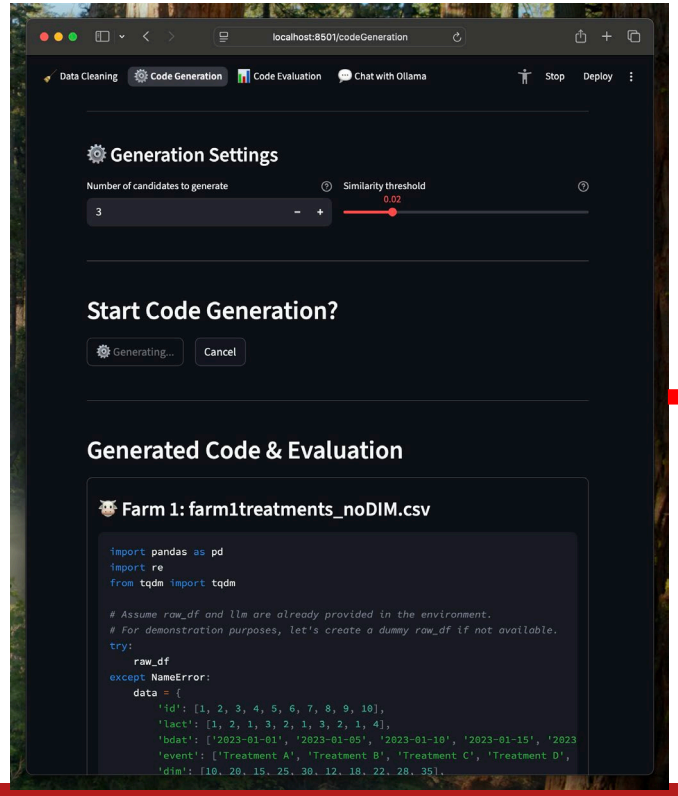
Web Interface

Before Farm

At Farm

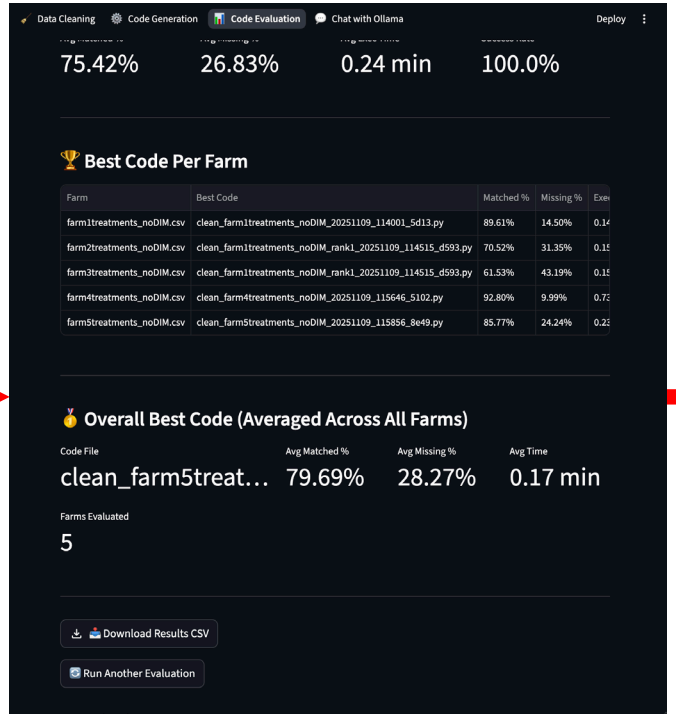
1. Generate code

Save best generations

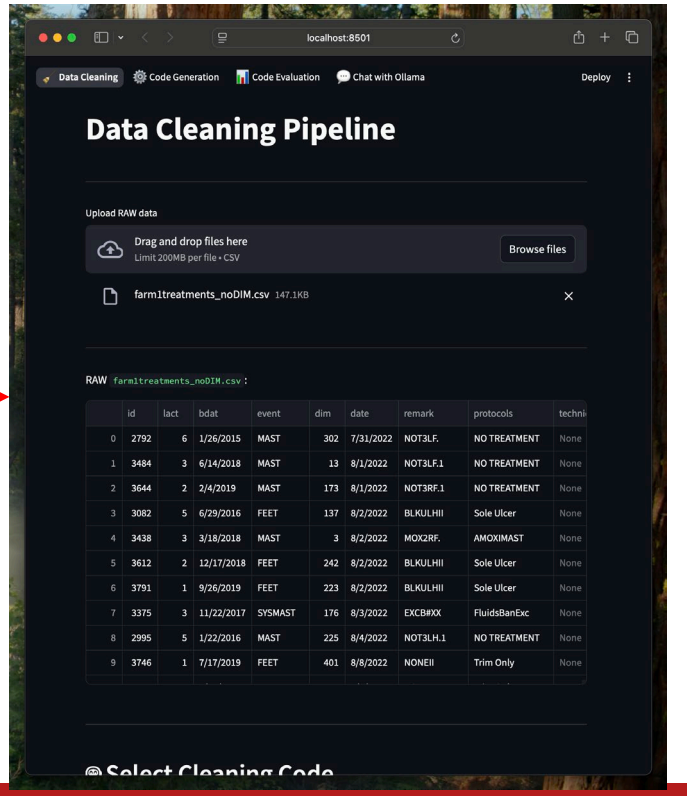


2. Evaluate

Rank all codes



3. Run best code on new farm



Experimental Design

```python
import pandas as pd
import re
import tqdm

# Note: The 'llm' variable is automatically provided at runtime.
# It is a LangChain chat model instance that you can use to invoke language model queries.
# Usage example:
#   prompt = "Your question here"
#   response = llm.invoke(prompt)
#   answer = response.content  # Extract the text response
```

- Simple task description
- Few-shot examples
  - For now, chosen randomly
- Dynamic LLM definition
  - Separates generation and runtime

```
### Example:
```
Protocol: ORBENIN.IMM
Remark: ORB/LOC
```

### Response:
yes

### Example:
```
Protocol: AMOXIMAST
Remark: MOX2RH
```

### Response:
yes

### Example:
```
Protocol: Sole Ulcer
Remark: BLKUQQII
```

### Response:
no
```

```python
def build_prompt(
    raw_df: pd.DataFrame,
    few_shot_path: Path = BASE_DIR / "few_shot.txt",
    predefined_path: Path = BASE_DIR / "predefined_func.txt",
) -> str:
    few_shot = few_shot_path.read_text()
    predefined_func = predefined_path.read_text()

    cols = ", ".join(raw_df.columns.to_list())

    return f"""
    You are a Python developer building an AI-powered data cleaning pipeline using a language model.

    You are working with a pandas DataFrame named `raw_df`, where each row represents a treatment event in a veteri
    Some of these events involve the use of **antibiotics**. The DataFrame contains the following columns: {cols}.
    Relevant context about the treatment is provided in the `Remark`, `Protocols`, and `Event` columns.

    Your task is to perform the following steps:

    0. clean the `raw_df` by any necessary preprocessing (e.g. remove whitespace in columns and values, handle miss
    - Convert the BDAT and Date to str (YYYY-MM-DD).
    1. Identify whether each treatment event involves the use of an **antibiotic** by querying a language model.
    - For each unique `Protocol`, construct a natural language prompt that asks whether the treatment involves anti
        - For each Protocol, the Remark might be different. But you can only use the first occurrence of Remark for
    - The prompt used in your code must explicitly instruct the model and be as detailed as possible, and make the
    - You can include the few-shot examples in your prompt - {few_shot} .
    - Use `tqdm` to display progress.
    2. Only keep the rows where the corresponding `Protocol` indicate the event involves antibiotic use (like there
    - Save the protocols that involve antibiotics in a dictionary named `antibiotic_protocols`, where the key is th
antibiotics.
    3. If the same `Protocols` name appears in multiple rows with different treatments (as indicated by differences
    use like `{{protocol}}_{{remark}}` to disambiguous them. If it is already unique, keep the original name.
    4. From the filtered DataFrame, keep only the columns: `['id', 'lact', 'bdat', 'event', 'dim', 'date', 'origina
       - The `original_protocol` column should contain the original protocol name before any disambiguation.
       - The `protocol` column should contain the disambiguated protocol name.
       Ensure that the column names is exactly the same as the columns above.

    Make the code as robust and efficient as possible, handling potential errors, exceptions and edge cases.

    Important:
        - `raw_df` is already loaded and available for use.
        - `llm` is a pre-configured LangChain chat model instance that is already available for you to use.
          You can invoke it with `llm.invoke(prompt)` where prompt is a string, and access the text response with `
          No imports are needed - the `llm` variable is automatically provided at runtime.
        - Use the following helper functions and example code patterns: {predefined_func}
        - Save final output as `cleaned_df`.

    Your code MUST be returned in a SINGLE continuous Python code block. Do not break it with long explanations. Al

    Return a **single Python code block** that completes this task.
    """
```

Results: Case Study

Example performant code

1. Preprocessing

```python
# 0. Clean the raw_df
# Make a copy to avoid modifying the original DataFrame if it's used elsew
df = raw_df.copy()

# Clean column names: remove leading/trailing whitespace
df.columns = df.columns.str.strip()

# Clean string values: remove leading/trailing whitespace and fill NaN wit
# 'bdat' and 'date' columns are handled separately for date formatting.
string_cols_to_clean = ['remark', 'protocols', 'event', 'technician']
for col in string_cols_to_clean:
    if col in df.columns:
        df[col] = df[col].astype(str).str.strip().replace('nan', '')

# Convert BDAT and Date to str (YYYY-MM-DD)
for col in ['bdat', 'date']:
    if col in df.columns:
        # Convert to datetime, coercing errors to NaT for invalid date for
        df[col] = pd.to_datetime(df[col], errors='coerce')
        # Format to YYYY-MM-DD string, NaT (Not a Time) values become NaN
        df[col] = df[col].dt.strftime('%Y-%m-%d').fillna('')
```

2. Generated LLM Prompt

Utilizes given examples

```python
protocol_antibiotic_status = {}
# Use tqdm to display progress for LLM calls
for protocol, row in tqdm(unique_protocols_df.iterrows(), total=len(unique_protocc
    remark = row['remark']

    # Construct the detailed prompt with few-shot examples
    prompt = f"""
    You are an expert in veterinary medicine and pharmacology. Your task is to det
    Carefully analyze the 'Protocol' and 'Remark' provided for each treatment.
    Respond with 'yes' if the treatment protocol explicitly indicates or strongly
    Respond with 'no' if the treatment protocol does not involve antibiotics, is a
    Your answer should be a single word: 'yes' or 'no'.

    ### Example:
    Protocol: ORBENIN.IMM
    Remark: ORB/LOC

    ### Response:
    yes

    ### Example:
    Protocol: AMOXIMAST
    Remark: MOX2RH

    ### Response:
    yes

    ### Example:
    Protocol: Sole Ulcer
    Remark: BLKUQQII

    ### Response:
    no

    ### Current Query:
    Protocol: {protocol}
    Remark: {remark}

    ### Response:
    """

    try:
        response = llm.invoke(prompt)
        answer = response.content.strip().lower()
        # Check for 'yes' in the answer, avoiding overly strict matching
        protocol_antibiotic_status[protocol] = 'yes' in answer
    except Exception as e:
        # Log the error and default to False for robustness
        print(f"Error querying LLM for Protocol '{protocol}': {e}")
        protocol_antibiotic_status[protocol] = False
```

3. Filter data & return desired information

```python
# Create the dictionary of protocols identified as antibiotics
antibiotic_protocols = {p: status for p, status in protocol_antibiotic_status.items() if

# Filter the original DataFrame based on identified antibiotic protocols
df_filtered_antibiotics = df[df['protocols'].isin(antibiotic_protocols.keys())].copy()


# 3. If the same Protocols name appears in multiple rows with different treatments
#    (as indicated by differences in Remark), use like `{protocol}_{remark}` to disambig
#    If it is already unique, keep the original name.

# Add 'original_protocol' column before any potential disambiguation
df_filtered_antibiotics['original_protocol'] = df_filtered_antibiotics['protocols']

# Prepare 'remark' column for disambiguation: fill NaN with empty string
df_filtered_antibiotics['remark_for_disambiguation'] = df_filtered_antibiotics['remark']

# Identify protocols that need disambiguation: those with more than one unique remark st
# (including empty string as a distinct remark).
ambiguous_protocols_series = df_filtered_antibiotics.groupby('original_protocol')['remar
ambiguous_protocols_set = set(ambiguous_protocols_series[ambiguous_protocols_series > 1]

def get_disambiguated_protocol(row):
    """
    Determines the disambiguated protocol name based on whether the original protocol
    is ambiguous (i.e., has multiple distinct remarks).
    """
    if row['original_protocol'] in ambiguous_protocols_set:
        # If the protocol is ambiguous, check its specific remark
        if row['remark_for_disambiguation'] == '':
            # If remark is empty, use the original protocol name (no suffix)
            return row['original_protocol']
        else:
            # If remark is not empty, append it for disambiguation
            return f"{row['original_protocol']}_{row['remark_for_disambiguation']}"
    else:
        # If the protocol is not ambiguous (only one unique remark or only one occurrenc
        # keep the original protocol name without modification.
        return row['original_protocol']

df_filtered_antibiotics['protocol'] = df_filtered_antibiotics.apply(get_disambiguated_pr


# 4. From the filtered DataFrame, keep only the columns:
#    ['id', 'lact', 'bdat', 'event', 'dim', 'date', 'original_protocol', 'protocol'].
final_columns = ['id', 'lact', 'bdat', 'event', 'dim', 'date', 'original_protocol', 'pro
cleaned_df = df_filtered_antibiotics[final_columns].copy()

# Final check to ensure 'bdat' and 'date' are string columns, as required.
for col in ['bdat', 'date']:
    if col in cleaned_df.columns:
        cleaned_df[col] = cleaned_df[col].astype(str)
```

Preliminary Results

Large variation between generations

Xinyu Yang

Top 3

Farm	Precision	Recall	Time (m)
farm1	0.95	0.87	0.45
farm2	0.96	0.94	0.48
farm3	0.99	0.91	0.3
farm4	1	0.7	1.52
farm5	0.78	0.76	2.06
all	0.94	0.84	0.96

Farm	Precision	Recall	Time (m)
farm1	0.95	0.81	0.61
farm2	0.96	0.94	1.08
farm3	0.99	0.91	1.1
farm4	0.84	0.72	1.66
farm5	0.76	0.72	2.16
all	0.9	0.82	1.32

Farm	Precision	Recall	Time (m)
farm1	0.95	0.8	0.09
farm2	0.96	0.94	0.09
farm3	0.99	0.91	0.1
farm4	0.86	0.76	0.11
farm5	0.78	0.76	0.13
all	0.91	0.83	0.1

Bottom 3

Farm	Precision	Recall	Time (m)
farm1	0.35	0.96	7.35
farm2	0.77	0.96	8.27
farm3	0.34	0.95	7.67
farm4	0.29	0.88	10.91
farm5	0.4	0.9	11.39
all	0.43	0.93	9.12

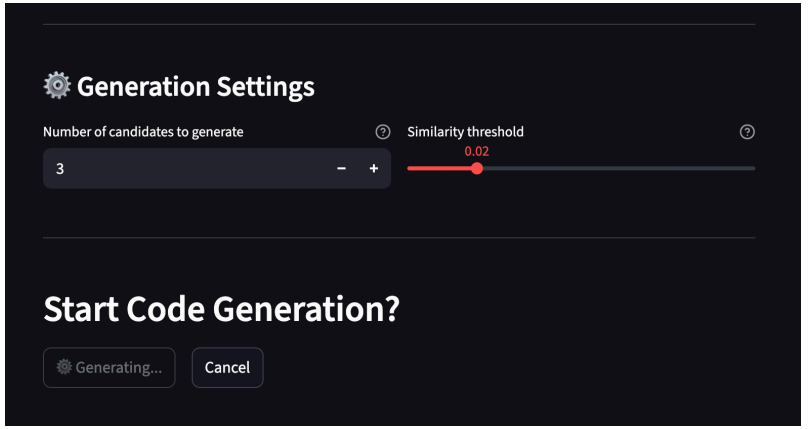
Farm	Precision	Recall	Time (m)
farm1	0.37	0.88	7.91
farm2	0.39	0.94	9.16
farm3	0.55	0.96	7.65
farm4	0.62	0.88	10.53
farm5	0.6	0.77	11.9
all	0.51	0.89	9.43

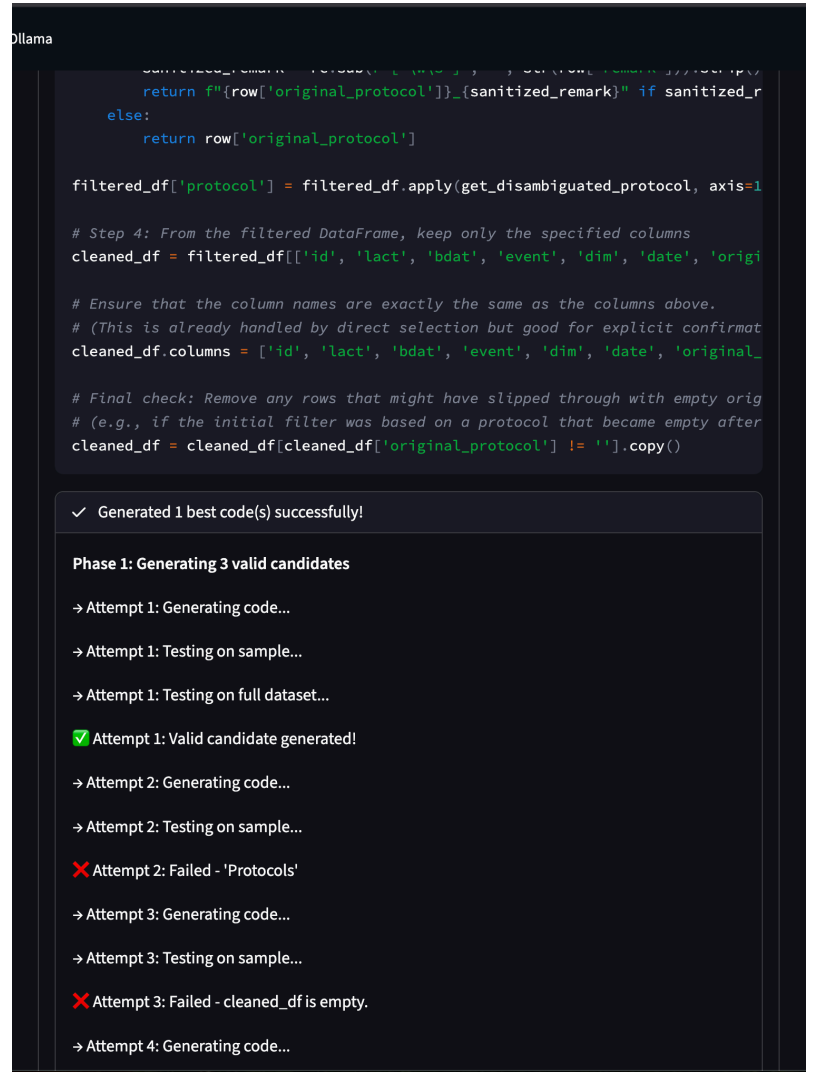
Farm	Precision	Recall	Time (m)
farm1	0.37	0.87	6.56
farm2	0.39	0.96	7.26
farm3	0.36	0.93	6.63
farm4	0.67	0.81	9.69
farm5	0.68	0.75	10.34
all	0.49	0.87	8.1

Web-based Evaluation

Worst generations are successfully pruned in generation phase

- Codes are generated in rounds of 3.
  - Simple validation ensures they can run on data without crashing.
- Once 3 are generated, only top performers are kept.
  - This helps eliminate poor performers at the generation step.

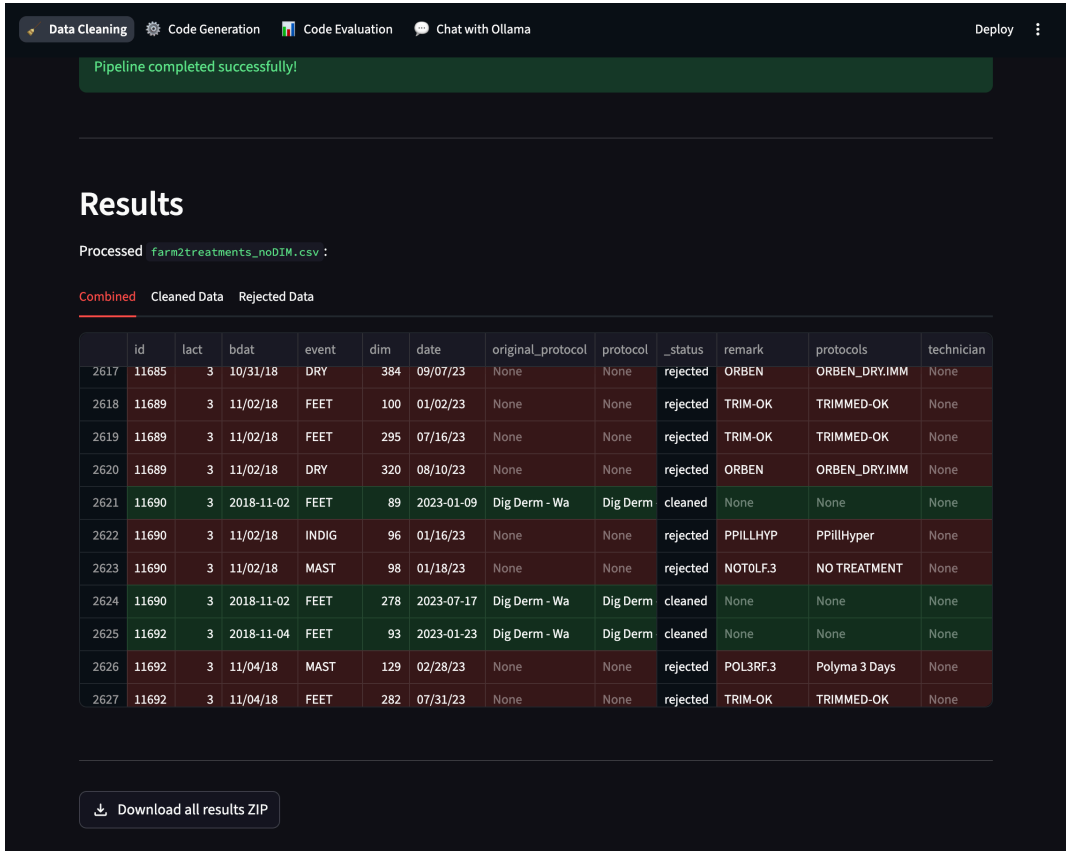




RESULTS

Conclusions

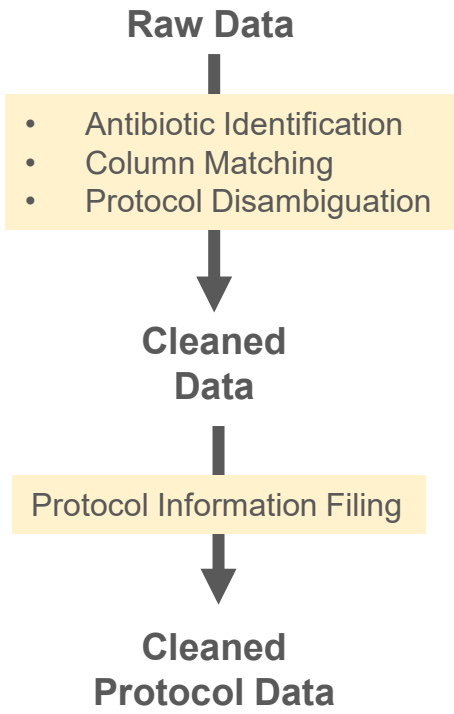
- LLM-based pipelines can standardize farm treatment data.
  - This can save significant human effort while maintaining accuracy.
  - Moving from manual to agentic workflows makes the system more flexible and scalable.
- Viable performance can be achieved without relying on cloud-based LLMs.
- Python code can easily be extended into a simple web interface.



	id	lact	bdat	event	dim	date	original_protocol	protocol	_status	remark	protocols	technician
2617	11685	3	10/31/18	DRY	384	09/07/23	None	None	rejected	ORBEN	ORBEN_DRY.IMM	None
2618	11689	3	11/02/18	FEET	100	01/02/23	None	None	rejected	TRIM-OK	TRIMMED-OK	None
2619	11689	3	11/02/18	FEET	295	07/16/23	None	None	rejected	TRIM-OK	TRIMMED-OK	None
2620	11689	3	11/02/18	DRY	320	08/10/23	None	None	rejected	ORBEN	ORBEN_DRY.IMM	None
2621	11690	3	2018-11-02	FEET	89	2023-01-09	Dig Derm - Wa	Dig Derm	cleaned	None	None	None
2622	11690	3	11/02/18	INDIG	96	01/16/23	None	None	rejected	PPILLHYP	PPillHyper	None
2623	11690	3	11/02/18	MAST	98	01/18/23	None	None	rejected	NOT0LF.3	NO TREATMENT	None
2624	11690	3	2018-11-02	FEET	278	2023-07-17	Dig Derm - Wa	Dig Derm	cleaned	None	None	None
2625	11692	3	2018-11-04	FEET	93	2023-01-23	Dig Derm - Wa	Dig Derm	cleaned	None	None	None
2626	11692	3	11/04/18	MAST	129	02/28/23	None	None	rejected	POL3RF.3	Polyma 3 Days	None
2627	11692	3	11/04/18	FEET	282	07/31/23	None	None	rejected	TRIM-OK	TRIMMED-OK	None

Next Steps

- Allow the agent to discover standardization rules (i.e. the prompt).
- Experiment with more advanced agent feedback.
  - Show the agent past, failed generations, and allow iteration.
  - Try replacing human-made evaluation data with pipeline results.
  - These changes require no modification to the website.
- Scale to new farms.



Acknowledgements

AUDIO Project

Renata Ivanek, DVM, MSc, PhD
Jennifer Sun, PhD
Katherine Koebel, DVM
Xinyu Yang
Daryl Nydam, DVM, PhD
Michael Capel, DVM
Ece Bulut, PhD

Funding

*This project is supported by the Food and Drug Administration (**FDA**) of the U.S. Department of Health and Human Services (HHS) as part of a financial assistance award U01FD008421 totaling $199,907 with 100 percentage funded by FDA/HHS and $0 amount and 0 percentage funded by non-government sources. The contents are those of the author(s) and do not necessarily represent the official views of, nor an endorsement, by FDA/HHS, or the U.S. Government. Research reported in this presentation was partially supported by the Office of the Director, National Institutes of Health (**NIH**) under Award Number T32ODO011000. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Center For Research Resources or the National Institutes of Health.*