Summary:

- 3 baselines, copy the value from last period, copy the value from the same period the year before, take the max between the last 2 periods
- Models on geo features alone can at best match the performance of the baseline
- Need to find a way to bring some improvement by adding conflict features!!

GKG data processing

In [ ]:
```python
from helper_functions.gdelt_data_mapping_optimized import load_gadm_data
from helper_functions.download_gdelt_gkg import (
    download_gkg_range,
    consolidate_gkg,
    process_gkg_data,
    consolidate_and_merge_fews_gkg,
    aggregate_files_by_day,
)
import multiprocessing
import pandas as pd
import glob
import os
import numpy as np

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 50)
pd.set_option('display.width', 1000)

# ---------------------------------------------------------------------
# Define paths
# ---------------------------------------------------------------------
gadm_path = "../data/gadm/gadm_410_filtered_v2.gpkg"
fews_path = "../data/fews/fews_with_conflicts_admin2.parquet"
gkg_raw_dir = "../data/gdelt/gkg/1_raw"
gkg_daily_dir = "../data/gdelt/gkg/1_raw_daily"
gkg_consolidated_dir = "../data/gdelt/gkg/2_consolidated"
gkg_mapped_dir = "../data/gdelt/gkg/3_mapped"
gkg_output_path = "../data/gdelt/gkg/4_aggregated/gkg_final.parquet"

os.makedirs(gkg_daily_dir, exist_ok=True)
os.makedirs(gkg_consolidated_dir, exist_ok=True)
os.makedirs(gkg_mapped_dir, exist_ok=True)
os.makedirs(os.path.dirname(gkg_output_path), exist_ok=True)

# ---------------------------------------------------------------------
# Step 1: Download & aggregate GKG
# ---------------------------------------------------------------------
# print("Downloading GKG data...")
# download_gkg_range(
#     start="2018-04-16",
#     end="2020-01-01",
#     raw_dir=gkg_raw_dir,
#     max_workers=8
# )

# aggregate_files_by_day(
#     input_folder=gkg_raw_dir,
#     output_folder=gkg_daily_dir,
#     start_date="20160101",
#     end_date="20240228"
# )

# ---------------------------------------------------------------------
# Step 2: Consolidate daily → monthly parquet
# ---------------------------------------------------------------------
# print("Consolidating GKG data...")
# consolidate_gkg(
#     raw_dir=gkg_daily_dir,
#     out_file=os.path.join(gkg_consolidated_dir, "files.parquet")
# )

# ---------------------------------------------------------------------
# Step 3: Load FEWS NET & GADM data
# ---------------------------------------------------------------------
fews_df = pd.read_parquet(fews_path)
fews_df = fews_df[['ADMIN0', 'ADMIN1', 'ADMIN2', 'period', 'CS_score']]

# print("Loading GADM data...")
# gadm_gdf, fews_df = load_gadm_data(gadm_path, fews_df)

# # ---------------------------------------------------------------------
# # Step 4: Combine consolidated monthly files
# # ---------------------------------------------------------------------
# month_files = glob.glob(os.path.join(gkg_consolidated_dir, "*.parquet"))
# if not month_files:
#     raise FileNotFoundError(f"No consolidated parquet files found in {gkg_

# gkg_df = pd.concat((pd.read_parquet(fp) for fp in month_files), ignore_inc
# print(f"Loaded {len(gkg_df):,} GKG records")

# # ---------------------------------------------------------------------
# # Step 5: Map to administrative regions
# # ---------------------------------------------------------------------
# print("Mapping administrative regions...")
# process_gkg_data(gkg_df, output_dir=gkg_mapped_dir)

# ---------------------------------------------------------------------
# Step 6: Merge with FEWS data
# ---------------------------------------------------------------------
print("Consolidating mapped GKG data with FEWS data...")
df_final = consolidate_and_merge_fews_gkg(mapped_gkg_dir=gkg_mapped_dir, few
print("Unique URLs:", len({str(u).strip() for x in df_final["DocumentIdentif
print(f"Final dataset rows: {len(df_final):,}")

for col in df_final.columns:
    if col.startswith("n_") or col in ["usd_aid"]:
        df_final[col] = pd.to_numeric(df_final[col], errors="coerce").fillna
```

```python
df_final.to_parquet(gkg_output_path, index=False)
print(f"Saved final GKG dataset to {gkg_output_path}")
```

In [6]:
```python
import pandas as pd
consolidated = pd.read_parquet("../data/gdelt/gkg/2_consolidated/gkg_2022_07
consolidated = consolidated[consolidated['V2Counts'] != 'None']
consolidated.head()
```

Out[6]:

	GKGRECORDID	DATE	SourceCollectionIdentifier	SourceCommonName	
6	20220729223000-1009	2022-07-29 22:30:00	1	gulf-times.com	
10	20220729014500-183	2022-07-29 01:45:00	1	catholicphilly.com	https
12	20220729014500-242	2022-07-29 01:45:00	1	dailymail.co.uk	
21	20220729014500-966	2022-07-29 01:45:00	1	nyasatimes.com	htt
23	20220729014500-1120	2022-07-29 01:45:00	1	wprl.org	http

5 rows × 44 columns

In [ ]:
```python
import pandas as pd
import numpy as np
import glob
import re
import os
from helper_functions.url_scraping_base import process_urls_in_chunks

# ----------------------------------------------------------------------
# Load GKG data
# ----------------------------------------------------------------------
gkg_path = "../data/gdelt/gkg/4_aggregated/gkg_final.parquet"
if not os.path.exists(gkg_path):
    raise FileNotFoundError(f"Missing file: {gkg_path}")

df = pd.read_parquet(gkg_path)
print(f"Loaded {len(df):,} GKG records from {gkg_path}")

# ----------------------------------------------------------------------
# Extract unique URLs from DocumentIdentifier
# ----------------------------------------------------------------------
unique_urls = sorted({
    str(u).strip()
    for arr in df["DocumentIdentifier"].dropna()
    for u in (arr if isinstance(arr, (list, tuple, np.ndarray)) else [arr])
    if isinstance(u, str) and u.startswith("http") and str(u).strip().lower(
})
print(f"Total unique URLs found: {len(unique_urls):,}")

# ---------------------------------------------------------------------
# Identify already processed URLs
# ---------------------------------------------------------------------
chunk_dir = "../data/gdelt/gkg/scraped_urls"
os.makedirs(chunk_dir, exist_ok=True)

parquet_files = glob.glob(os.path.join(chunk_dir, "chunk_*.parquet"))

chunk_numbers = (
    [int(re.search(r'chunk_(\d+)\.parquet', f).group(1)) for f in parquet_fi
    if parquet_files else []
)
highest_chunk = max(chunk_numbers) + 1 if chunk_numbers else 0
print(f"Highest existing chunk: {highest_chunk - 1 if highest_chunk > 0 else

if parquet_files:
    processed = pd.concat([pd.read_parquet(f) for f in parquet_files], ignor
    processed_urls = set(processed['url'].dropna().values)
else:
    processed_urls = set()

unique_urls = [u for u in unique_urls if u not in processed_urls]
print(f"Remaining URLs to process: {len(unique_urls):,}")

# ---------------------------------------------------------------------
# Run async URL scraping
# ---------------------------------------------------------------------
import nest_asyncio, asyncio
nest_asyncio.apply()

if unique_urls:
    process_urls_in_chunks(
        urls=unique_urls,
        chunk_size=10000,
        concurrency=150,
        chunk_id=highest_chunk,
        timeout=4,
        max_retries=3,
        max_selenium_workers=4,
        fallback_mode="async_only",  # no Selenium fallback
        output_dir=chunk_dir
    )
else:
    print("✅ All URLs already processed!")
```

Filter and clean urls

In [ ]:

```
import re
import os
import math
```

```python
import logging
import multiprocessing as mp
from collections import Counter

from tqdm.notebook import tqdm
import pandas as pd
import glob
import spacy

from helper_functions.topic_modelling.text_processing_parallel import prepro
from helper_functions.topic_modelling.flatten_articles import filter_article
from helper_functions.topic_modelling.deduplication import deduplicate_minha
from helper_functions.run_ner_parallel import run_ner_parallel, inject_count

# ============================================================
# 0️⃣ LOAD GKG SCRAPED ARTICLES
# ============================================================

parquet_dir = "../data/gdelt/gkg/scraped_urls"
parquet_files = glob.glob(os.path.join(parquet_dir, "chunk_*.parquet"))
if not parquet_files:
    raise FileNotFoundError(f"No parquet files found in {parquet_dir}")

df_scraped = pd.concat([pd.read_parquet(f) for f in parquet_files], ignore_i
print(f"Loaded {len(df_scraped):,} scraped GKG articles")

# ============================================================
# 1️⃣ PREPARE TEXT
# ============================================================

df_scraped["header"] = df_scraped["header"].fillna("").astype(str)
df_scraped["body"] = df_scraped["body"].fillna("").astype(str)
df_scraped["text"] = df_scraped["header"] + " " + df_scraped["body"]

# ============================================================
# 2️⃣ LEXICON FILTER
# ============================================================

print("Applying LEAP4FNSSA lexicon filter...")
df_scraped = filter_articles_by_lexicon(
    df_scraped,
    clean_text_col="text",
    lexicon_path="../data/LEAP4FNSSA_LEXICON_long.csv"
)
print("Remaining after lexicon filter:", df_scraped.shape)

# ============================================================
# 3️⃣ HEAVY NLP PREPROCESSING
# ============================================================

df_scraped = preprocess_text_parallel(df_scraped, text_col="text")
print("Text preprocessing done.")

# ============================================================
# 4️⃣ TRUNCATE CLEAN TEXT TO 500 WORDS
# ============================================================
df_scraped["clean_text"] = df_scraped["clean_text"].apply(
    lambda t: " ".join(str(t).split()[:500])
)

# ============================================================
# 5️⃣ DEDUPLICATION
# ============================================================

print("Running MinHash deduplication...")
df_scraped = deduplicate_minhash(df_scraped, text_col="clean_text", threshol
print("Remaining after dedup:", df_scraped.shape)

# ============================================================
# 6️⃣ NER EXTRACTION + DEMONYM INJECTION
# ============================================================

print("🔍 Running NER...")

df_scraped["clean_text"] = df_scraped["clean_text"].apply(inject_countries_f

n_process = 1 if "ipykernel" in mp.current_process().name.lower() else mp.cp
df_scraped = run_ner_parallel(df_scraped, text_col="clean_text", n_process=n

print("NER done.")

# Keep raw copies
df_scraped["NER_admin0_raw"] = df_scraped["NER_admin0"]
df_scraped["NER_admin1_raw"] = df_scraped["NER_admin1"]
df_scraped["NER_admin2_raw"] = df_scraped["NER_admin2"]

# ============================================================
# 7️⃣ LOAD FEWS COUNTRIES
# ============================================================

fews_path = "../data/fews/fews_with_conflicts_admin2.parquet"
fews_df = pd.read_parquet(fews_path)
fews_countries = [c.lower() for c in fews_df["ADMIN0"].unique()]

# ============================================================
# 8️⃣ REFINEMENT: PICK MOST FREQUENT FEWS COUNTRY PER ARTICLE
# ============================================================

def pick_main_country_from_text(text: str) -> str:
    if not isinstance(text, str):
        return None
    txt = text.lower()
    counts = Counter()
    for country in fews_countries:
        c = txt.count(country)
        if c > 0:
            counts[country] += c
    return counts.most_common(1)[0][0] if counts else None

def refine_ner_country(row):
    main = pick_main_country_from_text(row["clean_text"])
    if main:
        row["NER_admin0"] = main
    return row

df_scraped = df_scraped.apply(refine_ner_country, axis=1)
print("NER country refinement complete.")

# ============================================================
# 9️⃣ FILTER BY FEWS COUNTRIES
# ============================================================

df_scraped = df_scraped[
    df_scraped["NER_admin0"].isin(fews_countries) |
    df_scraped["NER_admin1"].isin(fews_countries) |
    df_scraped["NER_admin2"].isin(fews_countries)
]

print("Remaining after FEWS filter:", len(df_scraped))

# ============================================================
# 🔟 CRISIS FILTER (SAFE FOR GKG)
# ============================================================

# Removed "inflation/market/price/commodity" to avoid noise.
crisis_terms = re.compile(
    r"(famine|hunger|malnutrition|undernourish|food\s+crisis|food\s+security
    r"crop|harvest|yield|farmer|agricultur|seed|irrigat|fertiliz|pest|livest
    r"drought|flood|rainfall|storm|cyclone|heatwave|disaster|"
    r"aid|relief|humanitarian|"
    r"refugee|displaced|idp|"
    r"conflict|violence|clash|attack|insurgent|militia|war|unrest|protest)",
    re.I
)

df_scraped = df_scraped[
    df_scraped["clean_text"].str.contains(crisis_terms, na=False)
]

print("Remaining after crisis filter:", len(df_scraped))

# ============================================================
# 1️⃣1️⃣ SAVE OUTPUT
# ============================================================

out_path = "../data/gdelt/gkg/scraped_urls/cleaned_filtered_urls.parquet"
os.makedirs(os.path.dirname(out_path), exist_ok=True)

df_scraped.to_parquet(out_path, index=False)
print(f"Saved cleaned GKG dataset to: {out_path}")
```