Summary of DeepChronos Development

Executive summary: This report provides a detailed, step-by-step analysis of the development process for the "DeepChronos" forecasting system. The system was designed to merge quantitative time-series forecasting using the Chronos foundation model (via AutoGluon) with qualitative, human-like reasoning capabilities from the DeepSeek R1 Large Language Model (LLM). The overarching goal was to produce not only numerical predictions but also clear, context-aware explanations, akin to delivering both a quantitative projection and its accompanying analytical narrative. The development journey involved meticulous theoretical formulation, multiple implementation attempts fraught with specific errors, iterative code refinement addressing precise issues, strategic shifts in objectives to manage complexity and cost, and conscious efforts to frame the technical work using financial analogies. The final pipeline represents a novel tool for financial analysis, enhancing decision support through integrated, explainable forecasts, although its creation highlighted significant practical challenges related to computational resources, developer workload, and framework integration.

Detailed Evolution of the Forecasting Pipeline

The pipeline's construction unfolded through around 200 trial and error codes summarised in 13 runs, each documented with specific actions, errors, and resolutions:

Run 1: Initial Theoretical Pipeline Formulation

- **Objective:** To establish a rigorous, end-to-end mathematical model defining the proposed DeepChronos forecasting process.
- **Actions & Components:** The pipeline was defined through the following sequential steps:

    1. **Historical Window (Xt):**

$$(Xt)\colon\ Xt = \{xt - w + 1, \ldots, xt\}$$

This represents the input time series data, consisting of observations from time $t - w + 1$ up to time t, where w is the window length. This is a standard definition in time series analysis, providing the historical context for the forecast. The choice of w involves a trade-off between capturing sufficient history and computational burden/risk of non-stationarity.

2. **Chronos Tokenization (Tt):**

$$(Tt): \; Tt = \tau(Xt)$$

This critical step transforms the continuous (or discrete) time series window Xt into a sequence of tokens Tt,, suitable for input into a transformer-based model like Chronos. The function t encapsulates the tokenization strategy. Potential methods, though not specified in the summary, include:

- *Patching:* Dividing Xt into segments and embedding them.
- *Value Quantization:* Discretizing values into bins, using bin IDs as tokens. This tokenization process is fundamental for applying language model architectures to time series but inherently involves choices about information compression and representation granularity.

3. **Preliminary Forecast**

$$(y\sim t + 1): \; y\sim t + 1 = fC(Xt; \theta C).$$

Chronos (fC), parameterized by θC, generates an initial forecast $y\sim t + 1$ for the next step based on the historical window Xt. As a foundation model, Chronos likely leverages pre-training on vast amounts of time series data to achieve zero-shot or few-shot forecasting capabilities.

4. **Chain-of-Thought Embedding (Ct):**

$$(Ct): \; Ct = C(y\sim t + 1; \psi).$$

This step introduces the LLM component. It proposes generating an embedding Ct using a process C (parameterized by ψ) based on the preliminary Chronos forecast $y\sim t + 1$.. The "Chain-of-Thought" (CoT) designation strongly suggests that this embedding aims to capture structured reasoning or qualitative analysis about the forecast, likely generated by the DeepSeek-R1 model. This might involve prompting DeepSeek to analyze $y\sim t + 1.$ in context, and then embedding the resulting textual output (or intermediate reasoning steps) into the vector Ct.

5. **RL Refinement (DeepSeek-R1) (Δyt):**

$$(\Delta yt):\ \Delta yt = \pi(a|st;\theta RL), st = \{Tt, Ct\}$$

A Reinforcement Learning (RL) agent, represented by policy π (parameterized by θRL), is proposed to refine the forecast.

- *State (st):* Crucially, the state st incorporates both the tokenized raw history (Tt) and the CoT embedding (Ct). This allows the agent to base its decision on quantitative history and qualitative reasoning.
- *Action (a):* The action a is defined as the forecast adjustment Δyt.
- *Policy (π):* Maps the hybrid state st to the action Δyt. The goal is for the RL agent (implicitly guided by DeepSeek's capabilities via Ct) to learn optimal adjustments to the Chronos forecast.

6. **Final Forecast**

$$(y\hat{}t + 1):\ \hat{}t + 1 = y\sim t + 1 + \Delta yt.$$

The final prediction is the sum of the preliminary Chronos forecast and the RL-derived adjustment.

7. **Optimization Objective:**

$$min\theta C, \theta RL, \psi E[\sum t = 1T\gamma t\{(y\hat{}t + 1 - xt + 1)2 - \lambda I(Ct)\}]$$

This complex objective function aims to train the entire system end-to-end:

- *Joint Optimization:* Minimize the objective by adjusting parameters of Chronos (θC) the RL policy (θRL) and the CoT embedding process (ψ).
- *Components:* Includes an expectation over time/data (E), a discount factor (γt) a standard squared error term $(y\hat{}t + 1 - xt + 1)2$ for accuracy, and a novel term $\lambda I(Ct)$
- *Interpretability/Regularization Term ($I(Ct)$):* This term, modulated by λ, relates to the quality or information content of the CoT embedding Ct. It could penalize complexity or reward properties like semantic richness or alignment with true outcomes, balancing pure accuracy with the contribution of the LLM's reasoning.

Run 2: Initial Code Setup & Dependency Issues

- **Objective:** Set up the project environment and install necessary libraries.
- **Actions:**
  - Cloned the `open-r1` repository from GitHub.
  - Attempted installation of `flash-attn` using `pip install flash-attn --no-build-isolation`. This command bypasses PEP 517 build isolation, sometimes necessary for complex packages with specific build requirements. Encountered a build error requiring the `wheel` package, which was subsequently installed (`uv pip install wheel`). This highlights common issues with installing high-performance libraries that require compilation.
  - Attempted installing the main package in editable mode with development dependencies: `pip install -e ".[dev]"`. This revealed dependency conflicts and deprecation warnings, specifically mentioning `autogluon-multimodal 1.1.1` requiring `accelerate<0.22.0,>=0.21.0` but having `accelerate 0.30.1` installed, and `nltk` potentially interfering with `autogluon`. Managing conflicting dependencies between large frameworks (like AutoGluon) and core libraries (like Transformers/Accelerate) is a frequent challenge in complex Python projects. Editable mode (`-e`) is useful for development but can sometimes exacerbate dependency issues.

Run 3: Tokenizer Loading Issues (DeepSeek-R1)

- **Objective:** Load the tokenizer for the DeepSeek-R1 model (`open-r1-qwen-1.5b`).
- **Actions & Errors:** Several attempts failed:
  - `AutoTokenizer.from_pretrained("DeepSeek-R1/open-r1-qwen-1.5b")` resulted in a 401 Unauthorized error, indicating a need for authentication (likely a Hugging Face Hub token).

    - `AutoTokenizer.from_pretrained("open-r1/open-r1-qwen-1.5b")` resulted in a 404 Not Found error, suggesting an incorrect repository name or structure.
    - Loading from a local path (`./open-r1`) failed with `OSError: We couldn't connect... ensure your local path is correct and contains files listed in [...]`, specifically missing `config.json`. This indicates the local directory didn't contain the necessary configuration file defining the model architecture (`model_type`) expected by the `transformers` library.
    - **Resolution:** Successfully loaded the tokenizer using `AutoTokenizer.from_pretrained("./open-r1/open-r1-distill-qwen-1.5b", trust_remote_code=True)`. This implies the correct local path contained a subdirectory named `open-r1-distill-qwen-1.5b` which held the necessary files (`config.json`, tokenizer files, etc.), and `trust_remote_code=True` was required, likely because the model definition includes custom code.

Run 4: vLLM Server Setup & Querying (DeepSeek-R1)

- **Objective:** Serve the DeepSeek-R1 model using `vLLM` for efficient inference and query it.
- **Actions & Errors:**
    - Attempted to serve using `trl vllm-serve DeepSeek-R1/open-r1-distill-qwen-1.5b --tensor-parallel-size 1 --dtype bfloat16`. This failed because `bfloat16` precision requires GPU compute capability >= 8.0, while the available Tesla T4 GPU has capability 7.5. This highlights the hardware-dependency of certain numerical formats used for optimizing LLM performance.
    - **Resolution (Serving):** Successfully served the model by switching to `float16` precision: `trl vllm-serve ./open-r1/open-r1-distill-qwen-1.5b --tensor-parallel-size 1 --dtype half`. (`half` is often

synonymous with `float16`). It also noted successful startup logs indicating the use of `xformers` backend for optimized attention.

  - Attempted to query the server using `curl` with `http://localhost:8000/api/generate` and payload `{"prompt": "Implement QuickSort in Python:"}`. This failed with `{"detail":"Method Not Allowed"}`.
  - Attempted again with `http://localhost:8000/generate` and the same payload. This failed with `{"detail":[{"type":"missing","loc":["body","prompts"],"msg":"Field required",...}]}`.
  - **Resolution (Querying):** Successfully queried the server using the correct endpoint (`/generate`) and the correct JSON payload structure expected by vLLM's OpenAI-compatible endpoint: `curl http://localhost:8000/generate -H "Content-Type: application/json" -d '{"prompts": ["Implement QuickSort in Python:"], "max_tokens": 100}'`. The response contained token IDs, which require decoding using the previously loaded tokenizer to get human-readable text. This sequence demonstrates the necessity of precisely adhering to the API specification of the serving framework.

Run 5: Initial Forecasting Script Execution (V1)

- **Objective:** Run an initial version of the forecasting script (`V1deepchronos.py`).
- **Action & Error:** Executing the script resulted in a `KeyError: 'target'` during a "post-processing forecast results" step. This indicates the script expected a column named 'target' in a Pandas DataFrame or similar structure, but it was missing. This is a common issue when data formats don't match code expectations, often requiring data preprocessing or making the code more flexible.

Run 6: Benchmarking Strategy Discussion

- **Objective:** Discuss and plan the evaluation strategy for the DeepChronos model.

- **Discussion Points:**
  - **Frameworks:** Considered `FEV Eval` (lightweight, Hugging Face based, metrics like MASE, WQL) and `GIFT-Eval` (more comprehensive, resource-intensive). Alternatives like `sktime`, `GluonTS`, `Darts` were also mentioned as established libraries with evaluation capabilities.
  - **Scope:** Confirmed the possibility of restricting evaluation to specific dataset categories, such as "econ-fin" only.
  - **Compute Estimates:** Provided rough estimates for training/evaluation time:
    - Pre-training (Chronos/DeepSeek): Potentially days on multi-GPU setups (e.g., 8xA100).
    - Evaluation (FEV): Seconds to minutes per dataset on a single GPU (like Tesla T4/V100).
    - End-to-end Fine-tuning/RL (Hypothetical): Hours to days depending on complexity and data size. These estimates highlight the significant computational difference between large-scale pre-training and evaluating pre-trained models on specific tasks.

Run 7: FEV Benchmark Code Implementation

- **Objective:** Implement code to run the FEV benchmark using AutoGluon and Chronos.
- **Code Snippets & Logic:** Provided Python code demonstrating:
  - Importing necessary libraries (`FEV.Task`, `AutoGluonTabularPredictor`, `TimeSeriesDataFrame`, `TimeSeriesPredictor` from `autogluon.timeseries`).
  - Defining a task using `fev.Task(dataset="monash_tsf", name="tourism_monthly")`.
  - Loading data using `task.get_pandas()` which retrieves data from the Hugging Face Hub as Pandas DataFrames.

  - Converting Pandas DataFrames to AutoGluon's `TimeSeriesDataFrame` format.
  - Initializing `TimeSeriesPredictor` with settings like `prediction_length`, `eval_metric` (MASE), potentially specifying `hyperparameters={'Chronos': {}}` to use only Chronos.
  - Training the predictor using `.fit(train_data)`.
  - Generating predictions using `.predict(train_data)`.
  - (Implicitly) Evaluating predictions using `task.evaluate(predictions)`. This run laid out the basic structure for using FEV with AutoGluon.

Run 8: Debugging FEV Integration - Part 1 (Monash Tourism Monthly)

- **Objective:** Debug the FEV benchmark execution using the `monash_tsf/tourism_monthly` dataset.
- **Action & Error:** Running the code from Run 7 failed with `autogluon.timeseries.utils.data.RequiredColumnMissing`: "DataFrame is missing required columns: {'item_id'}".
- **Analysis:** The error clearly indicates that the `tourism_monthly` dataset, when loaded as a Pandas DataFrame via `task.get_pandas()`, does not contain the `item_id` column required by AutoGluon's `TimeSeriesDataFrame`. AutoGluon uses `item_id` to distinguish between different time series within the dataset.
- **Proposed Solution (and user constraint):** The initial suggestion was to manually add a default `item_id` column (e.g., `df['item_id'] = 'series_0'`). However, the user disallowed this, insisting on strictly following official examples which might handle this internally or expect data in a specific pre-processed format. This constraint highlights a common tension between pragmatic fixes and adherence to potentially opaque library internals or specific example workflows.

Run 9: Debugging FEV Integration - Part 2 (ETTm2)

- **Objective:** Debug FEV execution using the `ett/ETTm2` dataset.
- **Action & Errors:** Running the benchmark code on ETTm2 resulted in multiple errors:
    - `RequiredColumnMissing`: Missing `item_id` (same as Run 8).
    - `RequiredColumnMissing`: Missing `target`.
    - `ValueError: Please make sure that the timestamp column is compatible with`pd.to_datetime``.
    - `autogluon.timeseries.utils.data.TimeSeriesLengthRequirement`: "Time series in the dataset must contain at least 25 observations..." indicating some series were too short.
- **Analysis:** This dataset presented more format incompatibilities: lacking both `item_id` and the default `target` column (likely using a different name like 'OT'), potential issues with the timestamp format, and containing time series shorter than AutoGluon's default minimum length requirement for its models (including Chronos).

Run 10: Debugging FEV Integration - Part 3 (M4 Yearly)

- **Objective:** Debug FEV execution using the `m4/m4_yearly` dataset.
- **Action & Error:** Running the benchmark code on M4 Yearly failed with `pandas._libs.tslibs.np_datetime.OutOfBoundsDatetime: Out of bounds nanosecond timestamp: 2279-12-31 00:00:00` during the conversion to `TimeSeriesDataFrame`.
- **Analysis:** This points to timestamps in the dataset that fall outside the range representable by Pandas/NumPy's standard 64-bit nanosecond datetime format (roughly 1677 to 2262 AD). This can happen with synthetic data or data spanning very large historical or future periods. Handling requires either filtering such dates, using different date representations, or potentially custom handling within the library if supported.

Run 11: Refining FEV Integration Script (Addressing Runs 8-10)

- **Objective:** Develop a more robust script (`DeepChronos.py`) to handle the data inconsistencies encountered in Runs 8-10.
- **Key Script Enhancements:**
  - **FEV Version Check:** Added `assert fev.__version__ >= "0.3.1"`, important as library features/APIs change.
  - **Configuration:** Introduced YAML configuration (`config.yaml`) to specify datasets, target columns (`target_col`), and potentially Chronos model variants (`chronos_variant`). This makes the script more flexible and reusable.
  - **Flexible Data Loading:** Used `task.get_data(as_hf_dataset=True)` to load data as Hugging Face `Dataset` objects first, allowing for inspection and manipulation before conversion to Pandas/TimeSeriesDataFrame.
  - **Column Validation/Renaming:** Implemented explicit checks for required columns (`item_id`, `timestamp`, `target_col` from config) and added logic to rename columns if necessary (e.g., renaming the specified `target_col` to `target`). Added creation of a default `item_id` if missing.
  - **Timestamp Conversion:** Ensured robust conversion to datetime objects using `pd.to_datetime` with `errors='coerce'` to handle invalid dates (turning them into `NaT`) and subsequently dropping rows with `NaT` timestamps.
  - **Minimum Length Filtering:** Filtered out time series shorter than a minimum length threshold (e.g., 25) *before* fitting the predictor. This directly addresses the error seen in Run 9.
  - **Standardized Evaluation:** Ensured the FEV `task.evaluate(predictions)` method was called correctly.
  - **Error Handling & Logging:** Incorporated `try...except` blocks to catch errors during processing specific datasets and added logging (`logging` module) for better traceability.
  - **DeepSeek Integration Placeholder:** Included placeholders/comments indicating where the DeepSeek-R1

integration (querying the vLLM server with prompts based on Chronos forecasts) would occur.

Run 12: Further FEV Debugging & AutoGluon Configuration

- **Objective:** Continue debugging the refined script and correctly configure AutoGluon to use specific Chronos models.
- **Issues & Resolutions:**
    - **Dataset Loading:** Confirmed `task.get_data(as_hf_dataset=True)` was the correct approach.
    - **AutoGluon `fit` Parameters:** Discovered that `hyperparameters` to specify the model (e.g., `{'Chronos': {}}`) must be passed to the `.fit()` method, not the `TimeSeriesPredictor` constructor (`__init__`). Passing it to `__init__` might silently use default models instead.
    - **Model Naming:** Clarified correct model names like `'Chronos'` (for zero-shot) and identified that fine-tuned versions might require specific naming conventions (e.g., `'ChronosFineTuned[bolt_small]'` or similar, if available/registered within AutoGluon). Attempting to use a non-existent name like `'ChronosFineTuned'` would fail.
    - **Predictor Initialization:** Confirmed predictor initialization: `predictor = TimeSeriesPredictor(prediction_length=task.horizon, target="target", eval_metric=task.metric)`.
    - **Training Call:** Corrected training call: `predictor.fit(train_data, hyperparameters={'Chronos': {}})`.

Run 13: Integrating DeepSeek Server Launch and Management

- **Objective:** Integrate the launching and management of the DeepSeek vLLM server directly within the main Python script.

- **Challenges & Solutions:**
  - **Prompt Engineering:** Developed a more sophisticated prompt structure for DeepSeek, incorporating context like the Chronos forecast series, recent historical data, data frequency, forecast horizon, and evaluation metrics (MASE), aiming for richer textual explanations/refinements.
  - **Subprocess Launch:** Used `subprocess.Popen` to launch the `trl vllm-serve` command asynchronously.
  - **Environment Variables:** Needed to pass `CUDA_VISIBLE_DEVICES` correctly to the subprocess environment.
  - **Executable Path:** Encountered issues finding the `trl` executable. Solutions involved:
    - Using `shell=True` in `Popen` to rely on the system's PATH.
    - Finding the explicit path using `shutil.which('trl')` or manually specifying it if not in PATH.
  - **Process Management:** Implemented robust server startup checks (waiting for the server endpoint to become available) and shutdown procedures using `process.terminate()` and `process.kill()` within a `finally` block to ensure the server process is cleaned up even if errors occur.
  - **Long-Running Execution:** Recommended using `tmux` or `nohup` for running the entire script reliably in the background, preventing termination if the SSH session disconnects.

Results and Analysis

This section details the performance evaluation of the developed forecasting model (`ChronosFineTuned`, also referred to as DeepChronos) compared against a baseline model (`ChronosZeroShot`, specifically Chronos Bolt Base) using the FEV-Eval framework. The evaluation focused on time series forecasting tasks in general, using metrics derived from the FEV Leaderboard comparisons rather than purely financial data.

Evaluation Framework:

- The lightweight FEV-Eval tool was used for efficiency and to benchmark against established results on the FEV Leaderboard.

Model Comparison:

The primary comparison was between:

1. **`ChronosFineTuned` (DeepChronos):** The model developed in this project, involving fine-tuning on specific task data. It utilized the `ChronosFineTuned[bolt_small]` configuration based on previous debugging runs.
2. **`ChronosZeroShot` (Chronos Bolt Base):** A baseline using the pre-trained Chronos model without task-specific fine-tuning.

Performance Metrics:

Two key aspects were evaluated on representative tasks:

1. **Predictive Accuracy:** Measured by the validation score reported by AutoGluon, which corresponds to the negative of the Mean Absolute Scaled Error (-MASE). A higher score (less negative) indicates better accuracy (lower MASE).
    - **MASE Interpretation:** MASE compares the model's forecast error to the error of a naive baseline (e.g., predicting the last observed value). A MASE < 1.0 means the model is better than the naive baseline.
2. **Inference/Validation Runtime:** Measured in seconds, indicating the computational efficiency during the validation phase.

Quantitative Results:

The following table summarizes the performance on two specific Monash TSF datasets:

Task	Model	Validation Score (-MASE)	Implied MASE (Approx.)	Validation Runtime (seconds)
`monash_traffic`	`ChronosFineTuned`	-0.4148	0.4148	**7.48**
`monash_traffic`	`ChronosZeroShot`	-0.4413	0.4413	79.27

`monash_australian_electricity`	`ChronosFineTuned`	**-0.6868**	**0.6868**	**0.19**
`monash_australian_electricity`	`ChronosZeroShot`	-0.6296	0.6296	1.31

Export to Sheets

(Note: Lower MASE is better. Higher Validation Score (-MASE) indicates lower MASE, hence better accuracy. Lower Runtime is better.)

Key Observations from Results:

- **`monash_traffic` Task:**
  - `ChronosFineTuned` achieved slightly better predictive accuracy (MASE ≈ 0.4148) compared to `ChronosZeroShot` (MASE ≈ 0.4413).
  - `ChronosFineTuned` demonstrated a significantly faster validation runtime (7.48s vs 79.27s), over 10x faster.
- **`monash_australian_electricity` Task:**
  - `ChronosFineTuned` showed *worse* predictive accuracy (MASE ≈ 0.6868) compared to `ChronosZeroShot` (MASE ≈ 0.6296) on this specific task.
  - `ChronosFineTuned` maintained a considerably faster validation runtime (0.19s vs 1.31s), roughly 7x faster.

Integration Challenges and Caveats:

- **DeepSeek R1 Integration Status:** While Chronos was successfully integrated and fine-tuned (`Chronos Bolt Tiny` mentioned, likely referring to `bolt_small` used in code), challenges remain in obtaining *coherent output* from the DeepSeek R1 LLM component when processing Chronos results. There appears to be limited existing literature addressing this specific integration challenge.
- **Need for Further Fine-Tuning:** Although initial results are encouraging in some aspects (especially runtime), further fine-tuning is anticipated to significantly enhance overall model performance.
- **Project Constraints:** Resolving the DeepSeek output issues and performing additional optimization is likely to require more time and financial resources (beyond the approx. $40 USD already spent on Tencent Cloud) than available within the current project/thesis schedule.

Conclusion

The fine-tuned Chronos model (`ChronosFineTuned` / DeepChronos) demonstrates potential, particularly offering substantial improvements in computational efficiency (runtime) compared to the zero-shot baseline. Predictive accuracy results are mixed, showing slight improvement on one task but degradation on another relative to the baseline. Critical challenges remain in effectively integrating the reasoning component (DeepSeek R1) to achieve coherent explanations, and further optimization is needed, subject to time and budget constraints.

Code:

```
#!/usr/bin/env python3

"""

DeepChronos First-Iteration: Just Explain with ChronosFineTuned[bolt_small]


This pipeline:

  1. Reads tasks from `tasks.yaml` (with optional `target_col` per task).

  2. Cleans and standardizes each time-series DataFrame.

  3. Trains only ChronosFineTuned[bolt_small] via AutoGluon Chronos.

  4. Generates forecasts, builds a prompt, and queries DeepSeek R1 for a narrative
explanation.

  5. Prints each task's prompt and explanation.


Usage:

  python3 deepchronos_explain.py

"""


import os

import time

import logging

import yaml

import pandas as pd

import numpy as np
```

```
import requests
import pkg_resources
import subprocess

from autogluon.timeseries import TimeSeriesPredictor, TimeSeriesDataFrame
import fev  # pip install git+https://github.com/autogluon/fev.git@v0.3.1

# --------------------------
# Configure Logging
# --------------------------
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[logging.StreamHandler()]
)
logger = logging.getLogger("DeepChronos")

# --------------------------
# Utility: Standardize Columns
# --------------------------
def standardize_columns(df):
    df.columns = [col.lower() for col in df.columns]
    return df

# --------------------------
# Launch DeepSeek R1
# --------------------------
```

```
def start_deepseek_server():
    cmd = (
        "CUDA_VISIBLE_DEVICES=0 trl vllm-serve "
        "--model deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B --dtype=half"
    )
    env = os.environ.copy()
    try:
        process = subprocess.Popen(
            cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, env=env, shell=True
        )
        logger.info(f"Started DeepSeek R1 server with PID: {process.pid}")
        time.sleep(10)
        return process
    except Exception as e:
        logger.error(f"Failed to start DeepSeek R1: {e}")
        raise


# --------------------------
# Check fev Version
# --------------------------
def check_fev_version():
    try:
        version = pkg_resources.get_distribution("fev").version
        logger.info(f"Found fev version: {version}")
        parts = [int(x) for x in version.split('.')]
        if parts[0]<0 or (parts[0]==0 and parts[1]<3) or (parts[0]==0 and parts[1]==3 and
parts[2]<1):
            logger.warning("fev < 0.3.1, proceed with caution.")
```

```
            return False
        return True
    except Exception as e:
        logger.error(f"Failed to check fev version: {e}")
        return False

# --------------------------
# Load Tasks YAML
# --------------------------
def load_tasks_from_yaml(filepath="tasks.yaml"):
    with open(filepath) as f:
        tasks = yaml.safe_load(f)
    required = ['dataset_path','dataset_config','horizon','seasonality']
    valid = []
    for i, t in enumerate(tasks):
        missing = [r for r in required if r not in t]
        if missing:
            logger.warning(f"Task {i+1} missing {missing}, skipping.")
            continue
        valid.append(t)
    logger.info(f"Loaded {len(valid)} valid task(s)")
    return valid

# --------------------------
# Data Prep Functions
# --------------------------
def ensure_item_id(df):
```

```
    if 'item_id' not in df.columns:
        if 'id' in df.columns:
            df = df.rename(columns={'id':'item_id'})
        else:
            df['item_id'] = 1
    return df


def convert_timestamp(df):
    if 'timestamp' in df.columns:
        def fix(x): return x[0] if isinstance(x,(list,np.ndarray)) and len(x)>0 else x
        df['timestamp'] = df['timestamp'].apply(fix)
        df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
        df.dropna(subset=['timestamp'], inplace=True)
    return df


def convert_target(df):
    if 'target' in df.columns:
        df['target'] = pd.to_numeric(df['target'], errors='coerce')
    return df


def prepare_dataframe(df, target_col=None):
    df = standardize_columns(df)
    df = ensure_item_id(df)
    df = convert_timestamp(df)
```

```
    if target_col is None:
        target_col = 'target'
    if target_col not in df.columns:
        if 'ot' in df.columns:
            logger.info("Renaming 'ot' to target")
            df.rename(columns={'ot':target_col}, inplace=True)
        elif 'y' in df.columns:
            logger.info("Renaming 'y' to target")
            df.rename(columns={'y':target_col}, inplace=True)
        else:
            raise ValueError(f"Missing target column (expected '{target_col}' or 'ot' or 'y')")
    if target_col!='target':
        df.rename(columns={target_col:'target'}, inplace=True)
    df = convert_target(df)
    return df


# --------------------------
# Frequency Derivation
# --------------------------
def derive_frequency(seasonality):
    if seasonality in [24,48]: return 'H'
    if seasonality==12: return 'M'
    if seasonality==4: return 'Q'
    if seasonality==1: return 'D'
    if seasonality in [7,52]: return 'W'
    if seasonality in [365,366]: return 'D'
    logger.warning(f"Unknown seasonality {seasonality}, default 'D'")
```

```
    return 'D'


# -------------------------
# Train Chronos Fine-Tuned[bolt_small]
# -------------------------
def train_chronos_model(train_data, horizon, freq):
    model_path = f"chronos_model_{int(time.time())}"
    predictor = TimeSeriesPredictor(
        target='target',
        prediction_length=horizon,
        freq=freq,
        path=model_path,
        eval_metric='MASE'
    )
    logger.info(f"Training ChronosFineTuned[bolt_small] at {model_path}")
    predictor.fit(train_data, hyperparameters={'ChronosFineTuned[bolt_small]':{}})
    return predictor


# -------------------------
# Build Prompt
# -------------------------
def build_prompt(ts_id, forecast, last_obs, freq, horizon):
    return f"""
Time Series ID: {ts_id}
Frequency: {freq}
Prediction Horizon: {horizon}
Last Observed Values: {last_obs}
```

Forecasted Values: {forecast}

Explain the likely reasoning behind this forecast. What patterns or risks?""".strip()

```
# --------------------------
# Query DeepSeek
# --------------------------
def query_deepseek(prompt, url='http://localhost:8001/generate'):
    try:
        res = requests.post(url, json={'prompt':prompt}, timeout=60)
        res.raise_for_status()
        if 'application/json' in res.headers.get('content-type',''):
            data = res.json()
            return data.get('explanation') or data.get('response') or data.get('text') or
str(data)
        return res.text
    except Exception as e:
        logger.error(f"DeepSeek query failed: {e}")
        return f"Error: {e}"


# --------------------------
# Main Pipeline: Explain-Only
# --------------------------
def run_pipeline_explain_only():
    if not check_fev_version():
        logger.warning("Proceeding with caution: outdated fev version")
    tasks = load_tasks_from_yaml()
    results = {}
```

```
for idx, cfg in enumerate(tasks, start=1):
    name = cfg.get('dataset_config', f"task-{idx}")
    logger.info(f"[{idx}/{len(tasks)}] Processing {name}")
    try:
        task = fev.Task(
            dataset_path=cfg['dataset_path'],
            dataset_config=cfg['dataset_config'],
            horizon=cfg['horizon']
        )
        df_train, df_test, _ = fev.convert_input_data(task, adapter='pandas')
        df_train = prepare_dataframe(df_train, target_col=cfg.get('target_col'))
        df_test  = prepare_dataframe(df_test,  target_col=cfg.get('target_col'))
        train_ts = TimeSeriesDataFrame(df_train)
        test_ts  = TimeSeriesDataFrame(df_test)
        freq     = derive_frequency(cfg['seasonality'])
        pred_len = cfg['horizon']
        predictor = train_chronos_model(train_ts, pred_len, freq)
        fc = predictor.predict(test_ts).copy()
        col = 'mean' if 'mean' in fc.columns else 'forecast'
        if col not in fc.columns:
            other = [c for c in fc.columns if c not in ['item_id','timestamp']]
            col = other[0] if other else None
        if not col:
            raise ValueError('No forecast column found')
        fc.rename(columns={col:'forecast'}, inplace=True)
        first = fc.iloc[0]
        tsid = first['item_id']
```

```
            forecast_vals = fc[fc['item_id']==tsid].sort_values('timestamp')['forecast'].tolist()
            last_obs =
df_train[df_train['item_id']==tsid].sort_values('timestamp')['target'].tail(pred_len).tolist()
            prompt = build_prompt(tsid, forecast_vals, last_obs, freq, pred_len)
            explanation = query_deepseek(prompt)
            results[name] = {'prompt': prompt, 'explanation': explanation}
        except Exception as e:
            logger.error(f"Failed {name}: {e}")
            results[name] = {'error': str(e)}
    return results


# --------------------------
# Execute
# --------------------------
if __name__ == '__main__':
    ds_proc = start_deepseek_server()
    try:
        out = run_pipeline_explain_only()
        print("\n=== Pipeline Results ===")
        for cfg, res in out.items():
            print(f"\n-- {cfg} --")
            if 'error' in res:
                print("ERROR:", res['error'])
            else:
                print("Prompt:")
                print(res['prompt'])
                print("Explanation:")
                print(res['explanation'])
```

```
    finally:
        ds_proc.terminate()
        logger.info("DeepSeek server terminated.")
```

Results Directly from Terminal:

```
ubuntu@VM-0-16-ubuntu:~$ python3 main.py
2025-04-07 23:52:57,978 - ERROR - Failed to start DeepSeek R1 server: [Errno 2] No
such file or directory: 'CUDA_VISIBLE_DEVICES=0'
Traceback (most recent call last):
  File "/home/ubuntu/main.py", line 367, in <module>
    deepseek_process = start_deepseek_server()  # Start DeepSeek R1 server
  File "/home/ubuntu/main.py", line 50, in start_deepseek_server
    process = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
  File "/usr/lib/python3.10/subprocess.py", line 971, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "/usr/lib/python3.10/subprocess.py", line 1863, in _execute_child
    raise child_exception_type(errno_num, err_msg, err_filename)
FileNotFoundError: [Errno 2] No such file or directory: 'CUDA_VISIBLE_DEVICES=0'

ubuntu@VM-0-16-ubuntu:~$ python3 main.py
2025-04-07 23:57:44,204 - ERROR - Failed to start DeepSeek R1 server: [Errno 2] No
such file or directory: 'trl'
Traceback (most recent call last):
  File "/home/ubuntu/main.py", line 374, in <module>
    deepseek_process = start_deepseek_server()  # Start DeepSeek R1 server
  File "/home/ubuntu/main.py", line 57, in start_deepseek_server
    process = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, env=env)
  File "/usr/lib/python3.10/subprocess.py", line 971, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "/usr/lib/python3.10/subprocess.py", line 1863, in _execute_child
    raise child_exception_type(errno_num, err_msg, err_filename)
FileNotFoundError: [Errno 2] No such file or directory: 'trl'

ubuntu@VM-0-16-ubuntu:~$ CUDA_VISIBLE_DEVICES=0 trl vllm-serve --model
deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B --dtype=half
Command 'trl' not found, did you mean:
  command 'erl' from snap erlang (25.3)
  command 'erl' from deb erlang-base (1:24.2.1+dfsg-1ubuntu0.2)
  command 'trf' from deb trf (4.09.1-5)
  command 'trs' from deb konwert (1.8-13.2)
  command 'trn' from deb trn4 (4.0-test77-14)
```

command 'tre' from deb tre-command (0.3.6-2)
  command 'tdl' from deb devtodo (0.1.20+git20200830.0ad52b0-1)
  command 'tjl' from deb pvm-examples (3.4.6-3.2)
  command 'tbl' from deb groff-base (1.22.4-8build1)
  command 'tml' from deb tml (0.4.0-2ubuntu0.1)
ubuntu@VM-0-16-ubuntu:~$

ubuntu@VM-0-16-ubuntu:~$ cmd = "CUDA_VISIBLE_DEVICES=0 /usr/local/bin/trl vllm-serve --model deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B --dtype=half"
Command 'cmd' not found, but there are 20 similar ones.
ubuntu@VM-0-16-ubuntu:~$

ubuntu@VM-0-16-ubuntu:~$ tmux new -s mysession
python3 main.py
[exited]
2025-04-08 00:16:40,718 - INFO - Started DeepSeek R1 server with PID: 20456
2025-04-08 00:16:50,730 - INFO - Found fev version: 0.3.0
2025-04-08 00:16:50,730 - WARNING - fev version 0.3.0 is less than required 0.3.1
2025-04-08 00:16:50,730 - WARNING - fev version check failed; proceeding with caution.
2025-04-08 00:16:50,757 - INFO - Loaded 27 valid tasks from tasks.yaml
2025-04-08 00:16:50,757 - INFO - Processing task 1/27 with config: monash_traffic
2025-04-08 00:16:50,757 - INFO - Creating fev.Task with dataset_path=autogluon/chronos_datasets, config=monash_traffic, horizon=24

ERROR - Model training failed: At least some time series in train_data must have >= 25 observations. Please provide longer time series as train_data or reduce prediction_length, num_val_windows, or val_step_size.

ERROR - Task processing failed for monash_fred_md: At least some time series in train_data must have >= 25 observations.

--- Config: monash_traffic [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_australian_electricity [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: ercot [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: ETTm [failed] ---
Error: Following 1 columns are missing from the dataset: {'target'}. Available columns: ['id', 'timestamp', 'HUFL', 'HULL', 'MUFL', 'MULL', 'LUFL', 'LULL', 'OT']

--- Config: ETTh [failed] ---

Error: Following 1 columns are missing from the dataset: {'target'}. Available columns: ['id', 'timestamp', 'HUFL', 'HULL', 'MUFL', 'MULL', 'LUFL', 'LULL', 'OT']

--- Config: exchange_rate [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: nn5 [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_nn5_weekly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_weather [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_covid_deaths [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_fred_md [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: m4_quarterly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: m4_yearly [failed] ---
Error: Out of bounds nanosecond timestamp: 2279-12-31T12:00:00.000, at position 40

--- Config: dominick [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: m5 [failed] ---
Error: 'DataFrame' object has no attribute 'column_names'

--- Config: monash_tourism_monthly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_tourism_quarterly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_tourism_yearly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_car_parts [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_hospital [failed] ---

Error: No time series in train_data with >= 25 observations.

--- Config: monash_cif_2016 [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m1_yearly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m1_quarterly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m1_monthly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m3_monthly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m3_yearly [failed] ---
Error: No time series in train_data with >= 25 observations.

--- Config: monash_m3_quarterly [failed] ---
Error: No time series in train_data with >= 25 observations.

Following 1 columns are missing from the dataset: {'item_id'}. Available columns: ['id', 'timestamp', 'target']

ERROR - Task processing failed for monash_traffic: 'DataFrame' object has no attribute 'column_names'

Following 1 columns are missing from the dataset: {'item_id'}. Available columns: ['id', 'timestamp', 'target', 'subset']

CUDA_VISIBLE_DEVICES=0 trl vllm-serve --model deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B --dtype=half

PIPELINE RESULTS:

CONFIG: monash_traffic [status=failed]
 ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_australian_electricity [status=failed]
 ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: ercot [status=failed]

ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: ETTm [status=failed]
ERROR: Following 1 columns are missing from the dataset: {'target'}. Available columns: ['id', 'timestamp', 'HUFL', 'HULL', 'MUFL', 'MULL', 'LUFL', 'LULL', 'OT']

CONFIG: ETTh [status=failed]
ERROR: Following 1 columns are missing from the dataset: {'target'}. Available columns: ['id', 'timestamp', 'HUFL', 'HULL', 'MUFL', 'MULL', 'LUFL', 'LULL', 'OT']

CONFIG: exchange_rate [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: nn5 [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_nn5_weekly [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_weather [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_covid_deaths [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_fred_md [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: m4_quarterly [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: m4_yearly [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: dominick [status=failed]
ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: m5 [status=failed]
  ERROR: ID column item_id must contain unique values for each record

CONFIG: monash_tourism_monthly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_tourism_quarterly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_tourism_yearly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_car_parts [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_hospital [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_cif_2016 [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m1_yearly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m1_quarterly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m1_monthly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m3_monthly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m3_yearly [status=failed]

ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

CONFIG: monash_m3_quarterly [status=failed]
  ERROR: TimeSeriesPredictor.__init__() got an unexpected keyword argument 'hyperparameters'

{
  "completion_ids": [
    [
      3555, 646, 358, 653, 311, 1492, 498, 5267, 151649, 271, 9707, 0,
      358, 2776, 18183, 39350, 10911, 16, 11, 458, 20443, 11229, 17847,
      3465, 553, 18183, 39350, 13, 1752, 15817, 3565, 911, 1039, 4119,
      323, 3871, 11, 582, 21399, 498, 311, 8498, 1039, 3946, 9705, 13,
      151643
    ]
  ]
}

```
{
  "completion_ids": [
    [3555, 646, 358, 653, 311, 1492, …, 9705, 13, 151643]
  ]
}

Hello, I'm doing well today. Thank you for asking! How can
I assist you further?
```