WN–Wrangle: Wireless Network Data Wrangling Assistant

Anirudh Kamath
University of Utah
USA
anirudh.kamath@utah.edu

Dustin Maas
University of Utah
USA
dmaas@cs.utah.edu

Jacobus Van der Merwe
University of Utah
USA
kobus@cs.utah.edu

Anna Fariha
University of Utah
USA
afariha@cs.utah.edu

Abstract

Data wrangling continues to be the most time-consuming task in the data science pipeline and wireless network data is no exception. Prior approaches for automatic or assisted data-wrangling primarily target unordered, single-table data. However, unlike traditional datasets where rows in a table are unordered and assumed to be independent of each other, wireless network datasets are often collected across multiple measurement devices, producing *multiple, temporally ordered* tables that must be integrated for obtaining the complete dataset. For instance, to create a dataset of the signal quality of 5G cell towers within a geographic region, GPS data collected by cellphones must be joined with radio-frequency measurements of the corresponding cell towers. However, the join key `timestamp` typically exhibits mismatched sampling periods, causing a *misalignment*. Data-wrangling techniques for generic time-series datasets also fail here, since they lack knowledge of domain-specific data semantics—often defined by network protocols and system configurations. To aid in wrangling wireless network datasets, we demonstrate WN–Wrangle, an *interactive wrangling assistant*—tailored to the wireless network domain—that *suggests* the top-k next-best wrangling operations, along with rich, domain-specific *explanations*. Under the hood, WN–Wrangle enforces temporal-constraints- and a wireless-network-semantics-aware mechanism to score and rank an extended set of wrangling operators to improve the data quality. We demonstrate how WN–Wrangle identifies elusive data-quality issues specific to the wireless network domain and suggests accurate wrangling steps over datasets obtained from the widely used POWDER city-scale wireless testbed.

Link to demo video: https://users.cs.utah.edu/~afariha/wnwrangle.mp4

1 Introduction

The fifth generation (5G) of mobile networks is expected to serve nearly 3 billion users worldwide by 2026, with over 65% of fixed wireless access connections projected to be provided via 5G by 2025 [9]. Supporting this scale demands intelligent service monitoring, provisioning, and network planning, all of which increasingly rely on data-driven AI/ML techniques. For instance, Vodafone reported reducing data-ingestion latency from 36 hours to 25 minutes by leveraging insights from 70 petabytes of user-collected data [1]. However, a critical prerequisite for learning from such data is making it *analysis-ready*, which in practice requires extensive and routine *data wrangling* of large-scale wireless network (WN) datasets. Data wrangling, in general, is often a tedious and time-consuming process. Data scientists reportedly spend up to 60% of their time on tasks such as cleaning, imputing, transforming, and organizing data [7]. This has motivated the development of *assistant tools* for data wrangling (DW), such as CoWrangler [7] and Wrangler [12], which can suggest relevant operations to expedite the DW process.

However, existing general-purpose and context-agnostic DW assistants fall short when applied to WN datasets for several reasons. **First**, they assume that rows within a dataset are unordered and unrelated, often suggesting simple mean or mode imputation or even dropping rows with missing values. WN datasets, however, are inherently temporal, with measurements recorded sequentially or cyclically over time and often at fixed sampling rates. Applying generic imputation to such data can result in loss of temporal patterns or distort periodic signals. **Second**, generic DW tools typically operate on a single table at a time, which can fail catastrophically when multiple tables need to be joined, especially if they are misaligned [6] due to different measurement periods. **Third**, existing DW assistants are domain-agnostic and ignore crucial semantics of WN data. For instance, logarithmic units such as decibels must be converted to linear units before aggregation. We proceed to highlight these issues in Example 1.

Example 1. *Naomi is collecting data to train an intelligent network configuration management system (Figure 1). This system requires per-second, complete radio frequency (RF) measurements across all frequency channels, with corresponding GPS coordinates. She uses two devices for data collection: an RF measurement device that records data in the format* `<timestamp, frequency, RSRP>`[1] *as shown in* $\boldsymbol{A}$*, and a smartphone that records data in the format* `<timestamp, latitude, longitude>` *as shown in* $\boldsymbol{B}$*. Her goal is to generate a dataset in the schema* `<timestamp, frequency, RSRP, latitude, longitude>` *(shown in* $\boldsymbol{D}$*) by joining* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *over the join key* `timestamp`*.*

However, while trying to join $\boldsymbol{A}$ *and* $\boldsymbol{B}$*, Naomi observes that the measurements are logged at different temporal granularities:* $\boldsymbol{A}$ *contains multiple entries within a second while* $\boldsymbol{B}$ *has missing entries for some seconds. Furthermore, there is no exact match between the* `timestamp` *attributes of the two tables. To fix these issues, Naomi must temporally align the two datasets. Existing wrangling assistants like CoWrangler [7] cannot help Naomi here, rather, it exacerbates the situation by providing incorrect suggestions: (1) It suggests dropping the row* a_7 *from* $\boldsymbol{A}$ *due to the missing* `RSRP`*, which results in losing the only entry for the corresponding timestamp and frequency. This is due to CoWrangler not considering Naomi's goal—performing a join between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$*. (2) CoWrangler further suggests imputing* a_7`[RSRP]` *with the arithmetic mean of the attribute, which is incorrect as well, because the unit for* `RSRP` *is logarithmic (decibel-milliwatts).*

To join $\boldsymbol{A}$ *and* $\boldsymbol{B}$*, Naomi must ensure that (1) each* `frequency` *channel has exactly one* `RSRP` *reading per second, as in* $\boldsymbol{A_w}$*; (2) exactly one geo-location entry exists per second in the GPS data, as in* $\boldsymbol{B_w}$*; and (3) the values in* `timestamp` *in* $\boldsymbol{A}$ *&* $\boldsymbol{B}$ *match exactly. To achieve this, the following wrangling steps are essential:*

[1]RSRP (Reference Signal Received Power) measures the power of the reference signal received by a device from a cell tower, typically expressed in dBm, a logarithmic unit.

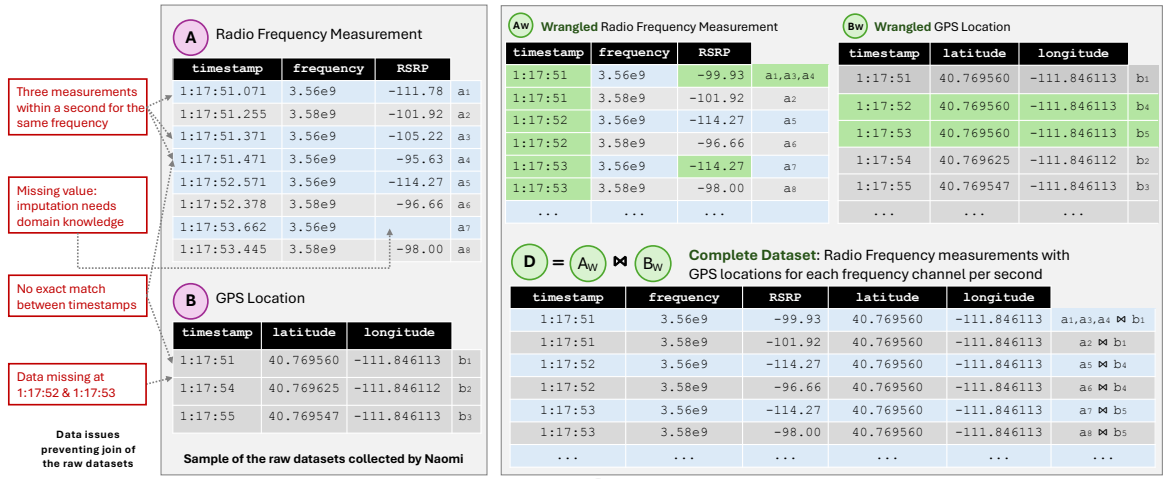

Figure 1: A: RF measurement data sample, B: GPS data sample, A_w & B_w: desired wrangled datasets, D: desired complete dataset.

***Step 1**: Forward-fill the missing* RSRP *in* a_7 *using the value from* a_5*, but not* a_6 *since the frequency channels are different.*
***Step 2:** (a) Merge rows* a_1*,* a_3*, and* a_4 *using techniques appropriate for logarithmic units—which ensures a per-second periodicity in* A_w *—and (b) round down timestamps to the nearest second.*
***Step 3:** (a) Insert two empty rows between* b_1 *and* b_2 *to fill the missing seconds and achieve a per-second periodicity in* B_w*, followed by (b) forward-fill the location values in the newly inserted rows, because smartphones suspend GPS logging while stationary to conserve power.*

Naomi had to spend 20 minutes writing and validating the mundane code snippet below—a disproportionate time cost despite her domain expertise—to obtain the desired dataset $D = A_w \bowtie B_w$.

```
# (Step 1) Forward-fill RSRP per frequency
A["RSRP"] = A.groupby("frequency")["RSRP"].ffill()

# (Step 2) Per-second log-aware RF aggregation
A["sec"] = A["timestamp"].dt.floor("S")
A["RSRP_lin"] = 10**(A["RSRP"]/10)
A = A.groupby(["sec", "frequency"]).agg({"RSRP_lin" : "mean"})
A["RSRP"] = 10*A["RSRP_lin"].apply(log10)
A.drop(columns = "RSRP_lin", inplace=True)

# (Step 3) Per-second GPS forward-fill
B = B.set_index("timestamp").asfreq("1S").ffill().reset_index()
```

A wrangling assistant for wireless network data. Example 1 demonstrates several wrangling operations that are specific to WN domain and are typically overlooked by generic wrangling assistants. This observation motivates the need for a specialized WN wrangling assistant that can proactively *suggest* domain-relevant wrangling operations. Such an assistant must satisfy the following requirements: (i) respect temporal constraints, such as ensuring periodicity and completeness (Step 1, Step 2 (a), & Step 3); (ii) support automatic alignment across tables through downsampling (Step 2 (a)), upsampling (Step 3 (a)), and homogenization (casting timestamp to the nearest second in Step 2 (b)); (iii) exploit inter-row relationships, including appropriate imputation strategies (forward-filling in Step 1 and Step 3 (b)); and (iv) preserve domain-specific semantic correctness, such as using imputation methods suitable for logarithmic units (Step 2 (a)). Finally, to encourage broad adoption, a WN wrangling assistant should additionally provide (v) rich *explanations* for its suggested operations; and (vi) *interactive* controls that allow users to inspect, customize, and guide the wrangling process.

We demonstrate WN–Wrangle, an interactive **W**ireless **N**etwork data-**Wrangl**ing assistant that suggests top-k relevant wrangling operations in a multi-table setting, tailored for WN datasets, satisfying the above six requirements. The key idea behind WN–Wrangle is *temporal-constraint*-aware scoring of *WN-specific* wrangling operations, which builds on the notion of temporal functional dependencies [5] to identify periodicity violations in the data.

Related work. Prior works in the databases community [7, 12] mainly target unordered relational datasets and thus fail on WN datasets, which are temporal in nature. Tools for cleaning temporal data using temporal integrity constraints [5] assume that domain-specific rules can be discovered from the data itself, which does not hold for the WN domain, where rules are often dictated by network protocols or system configurations. Recent works on time-series data wrangling [13] are domain-agnostic and fail to provide alignment-related wrangling suggestions such as upsampling, downsampling, or homogenizing (as shown in Example 1). While wrangling code generation systems [11] and foundation models have achieved partial success due to their ability to understand data semantics, they still struggle with complex tasks such as joining multiple incomplete datasets with misaligned join keys. For instance, when asked to join the datasets **A** and **B** while ensuring one record per second, ChatGPT made several mistakes, including incorrect RSRP imputation, incorrect frequency aggregation, and failing to up/downsample [2]. Commercial tools for time-series data [8] primarily target enterprise data sources and require extensive manual configuration.

Demonstration. In our demonstration, participants will observe how WN–Wrangle identifies temporal inconsistencies in two real-world datasets collected using the POWDER [10] city-scale wireless testbed, and suggests accurate wrangling steps in an explainable and interactive manner. We will showcase how a user can customize the suggestions in two WN data analysis scenarios—one to improve the readability of radio-frequency readings on a map interface, and the other to improve an ML model's performance on the collected data.

We provide an overview of WN–Wrangle's inner working in Section 2, and a walkthrough of the demonstration scenario based on Example 1 in Section 3.

2 System overview

WN–Wrangle targets multiple WN tables that share a context that allows their joint use via join or union. While we focus on the temporal context in this demonstration, WN–Wrangle supports other contexts such as frequency channels, spatial proximity, or device identity, by leveraging the corresponding alignment semantics.

Challenges. We identify the following key challenges towards building WN–Wrangle: **(C1)** How to *model WN-specific constraints* and *detect their violations* in the data? **(C2)** Which WN-specific wrangling operations should be considered as *candidates* for repairing the constraint violations? **(C3)** How to quantify the *effectiveness* of candidate wrangling operations—i.e., score and rank them—to enable accurate top-k suggestions automatically? **(C4)** How to generate domain-aware *explanations* for the suggestions, while allowing users to (optionally) guide the wrangling process *interactively*?

Overview. To address these challenges, WN–Wrangle comprises five components: (§2.1) a *semantic profiler* that profiles the data attributes; (§2.2) a *constraint discovery module* that discovers WN-specific constraints; (§2.3) a *Domain Specific Language (DSL)* tailored towards WN data; (§2.4) a *scoring method* that evaluates effectiveness of candidate wrangling operations; and (§2.5) an *explanation module* that generates explanations for the suggestions.

2.1 ***Semantic profiler.*** WN–Wrangle must understand the data characteristics to model WN-specific constraints (**C1**), support scoring of wrangling operations (**C3**), and provide explanations (**C4**). To this end, WN–Wrangle employs a *semantic profiler* that analyzes each data attribute over a small sample to infer the following:

- its data type (e.g., numerical, categorical, ordinal);
- its semantic type, measurement unit, and scale (e.g., `RSRP` measures signal power in `dBm`, a logarithmic scale);
- semantically correct aggregation strategies (e.g., converting to linear scale before averaging logarithmic values);
- semantic-type- and domain-aware imputation strategies (e.g., forward-filling GPS locations); and
- semantic dependencies between attributes (e.g., `RSRP` is typically measured per `frequency` channel).

To obtain the above information, WN–Wrangle queries a general-purpose LLM (GPT-5), eliminating the need for manual input from a domain expert. While a general-purpose LLM suffices in most cases, a fine-tuned variant trained on WN-specific documents and manuals can further improve the semantic profiler.

2.2 ***Constraint discovery module.*** To address **C1**, this module models WN-specific data constraints (e.g., temporal constraints requiring per-second measurements) and detects their violations in the data. Since enforcing these constraints ensures temporal and informational completeness, they provide guidance for suggesting wrangling operations that reduce constraint violations. To model temporal constraints, we use the established notion of temporal functional dependencies (TFDs) [5]. For example, TFD_1 = [Δ, `frequency`]$\rightarrow$ `RSRP` denotes the temporal constraint of having "exactly one `RSRP` record per second per `frequency` channel", where Δ = `1s` denotes a fixed periodicity in `timestamp`.

Discovering temporal constraints. WN–Wrangle discovers TFDs and associated parameters by leveraging semantic data profiles—which provide domain knowledge such as RF measurements require non-empty `RSRP` per second —and by analyzing the temporal periodicities in the data tables under consideration. Specifically, WN–Wrangle identifies a common periodicity Δ that can be achieved across the data tables without impacting too many tuples (minimizing side-effects). While TFD discovery [5] is fully automated, WN–Wrangle supports optional user input to validate Δ and to specify custom TFDs, thereby addressing **C4**.

Detecting violating tuples. WN–Wrangle flags tuples within the inferred periodicity that violate TFDs. E.g., in Figure 1, a_1, a_3, and a_4 violate TFD_1 due to multiple `RSRP`, while a_7 due to missing `RSRP`.

2.3 ***DSL for WN.*** To address **C2**, we need a domain-specific language that includes common operators used in WN data analysis. In this demo, we include frequently used operators identified through analyzing scripts from members of the POWDER [10] team.

(1) **Upsample**: Achieves the given temporal granularity by inserting new rows and filling in empty cells (Step 3 of Example 1).
(2) **Downsample**: Achieves the given temporal granularity by aggregating existing rows (Step 2 of Example 1).
(3) **Impute**: Imputes a cell using the given technique such as forward-fill or backward-fill (Step 1 of Example 1).
(4) **Round**: Rounds timestamps to the nearest temporal boundary (e.g., a whole second), based on the specified periodicity (Step 2).
(5) **Drop row**: Drops rows based on the given conditions.

Remark: The above DSL is specific for handling WN-data issues and are insufficient to resolve generic issues such as formatting inconsistencies, for which a generic DW tool [7, 12, 13] can be used.

2.4 ***Scoring method.*** WN–Wrangle generates a set of candidate operations by parameterizing the DSL operators with appropriate parameters. During this step, WN–Wrangle prunes semantically invalid candidates, i.e., those that violate WN-specific semantics, such as using an arithmetic mean as an imputation technique for `RSRP`. While this reduces the search space, a key challenge (**C3**) remains: determining which of these candidates should be suggested to the user. To this end, WN–Wrangle employs an efficient scoring method that estimates the expected improvement in data quality—i.e., the reduction in constraint violations—if a candidate wrangling operation were applied to the data.

Given a dataset D, discovered constraints or rules R, semantic profiles S, candidate wrangling operations W, and a violation function $V(D, R)$ that computes the degree of violation by D w.r.t R, the scoring method applies each $w \in W$ to a small sample of the data D' to obtain D'_w, where the sampling technique used is aware of the WN-specific semantics. This enables WN–Wrangle to efficiently estimate the expected data-quality improvement when w is applied over D as $V(D', R) - V(D'_w, R)$. While minimizing constraint violations is the primary objective, WN–Wrangle also accounts for data side effects by ensuring that each of the k suggested wrangling operations satisfy a predefined data-side-effect budget (e.g., at most p% cells/rows of the data can be modified).

2.5 ***Explanation module.*** Finally, the explanation module combines the information from the semantic profiler, constraints discovered and violations detected by the constraint discovery module, and the estimated reduction in violation obtained by the scoring method to generate a human-understandable, natural-language explanation for each of the suggested wrangling operations.

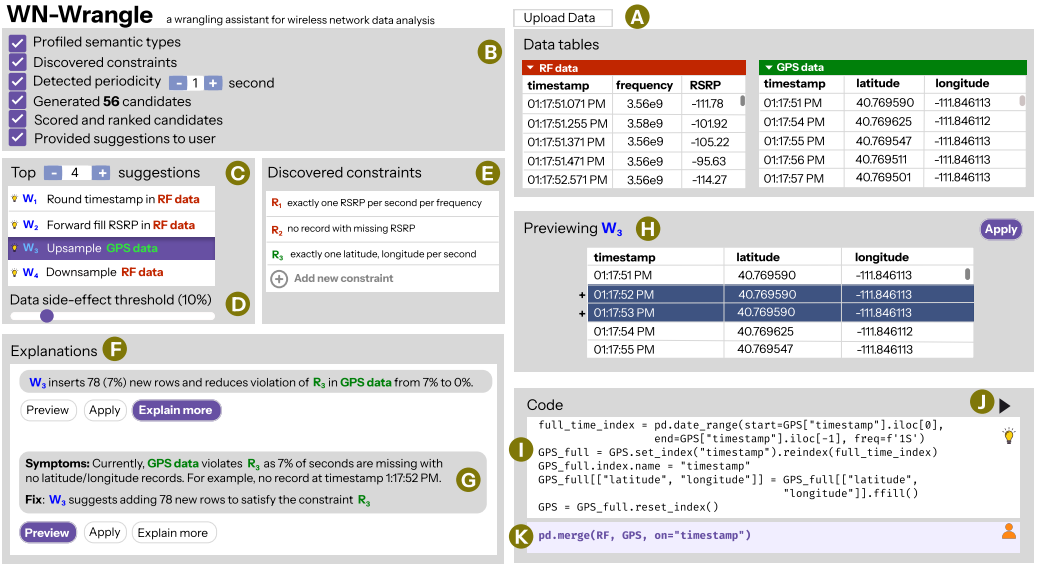

Figure 2: WN–Wrangle interface: (A) data upload and preview; (B) progress tracker for the WN–Wrangle workflow; (C) suggested wrangling operations; (D) user-specified threshold on data side effects; (E) discovered constraints; (F) explanations of the suggestions with interactive support; (G) follow-up clarifications; (H) on-demand preview of a selected suggestion; (I) editable code synthesized by WN–Wrangle for the selected suggestion; (J) execution button to apply the suggestion to the full dataset; (K) custom user code for joining the wrangled tables.

3 Demonstration

We will demonstrate WN–Wrangle over real-world POWDER datasets [3]. We will guide users through eleven steps of Figure 2 impersonating Naomi over the dataset [4] of Example 1.

Step (A) **(Data upload and preview)**. The user uploads two data files `RF.csv` and `GPS.csv` and previews the data. WN–Wrangle displays the first five rows. The user can scroll to see more.

Step (B) **(Workflow progress tracker)**. WN–Wrangle semantically profiles the data attributes (§2.1); discovers temporal and other constraints and detects the periodicity parameter Δ (§2.2), which the user can refine if they wish to; generates 56 wrangling candidates, scores, and ranks them (§2.4) to generate the suggestions.

Step (C) **(Wrangling suggestions)**. WN–Wrangle suggests 4 operations: W_1, W_2, & W_4 for the `RF` data and W_3 for the `GPS` data.

Step (D) **(Data side-effect)**. Along with the suggestions, WN–Wrangle displays maximum data side-effect incurred (10% rows were impacted) by any of the suggestions. The user can adjust the side-effect threshold and WN–Wrangle ensures that each suggested operation satisfies the user-specified side-effect requirement.

Step (E) **(Discovered constraints)**. WN–Wrangle lists the constraints (three in this case) that it used to compute the degree of violation for scoring the candidate wrangling operations. For example, the constraint R_1: "*exactly one* `RSRP` *record per second per frequency*" applies to the `RF` data. The user can add new constraints.

Steps (F) & (G) **(Explanation and interaction)**. The user wants to understand the rationale behind the suggestion W_3. WN–Wrangle provides an initial explanation—"W_3 *inserts 7% new rows, satisfying* R_3"—along with options to *Preview* its impact, *Apply* W_3 to the full data, or request further *Explanation*. After selecting "*Explain more*," WN–Wrangle offers a detailed explanation describing missing readings across multiple seconds. Satisfied, the user chooses to *Preview* the impact of W_3 on the `GPS` data.

Step (H) **(Preview suggestion impact)**. WN–Wrangle previews the impact of W_3 on a small sample of the `GPS` data, highlighting newly inserted rows by "+". Satisfied, the user clicks on "Apply".

Steps (I) & (J) **(Automatically generated wrangling code)**. WN–Wrangle inserts an editable code snippet for W_3 to the user notebook, and automatically executes it in Step (J). The user accepts the other three suggestions in a similar way (not shown).

Step (K) **(Custom code)**. Finally, the user successfully joins the two (now wrangled) datasets as per Example 1.

While WN–Wrangle applies to any WN data scenario, it is especially useful for experimental wireless testbeds like POWDER [10], where hundreds of users generate diverse datasets from 2,000+ yearly experiments, supporting advanced wireless applications.

References

[1] 2022. Vodafone. https://tinyurl.com/VodafoneNews.
[2] 2026. https://chatgpt.com/share/6962d019-1310-8009-a14d-a18306a987e6.
[3] 2026. Powder Data Sets. https://powderwireless.net/data.
[4] 2026. RF and GPS Dataset. https://powderwireless.net/data#2026-cell-metrics.
[5] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. 2015. Temporal rules discovery for web data cleaning. *PVLDB* 9, 4 (2015).
[6] A. Alkhateeb, G. Charan, T. Osman, A. Hredzak, J. Morais, U. Demirhan, and N. Srinivas. 2023. DeepSense 6G: A Large-Scale Real-World Multi-Modal Sensing and Communication Dataset. *IEEE Communications Magazine* 61, 9 (2023).
[7] B. Chopra, A. Fariha, S. Gulwani, A. Z. Henley, D. Perelman, M. Raza, S. Shi, D. Simmons, and A. Tiwari. 2023. CoWrangler: Recommender System for Data-Wrangling Scripts. In *SIGMOD 2023*.
[8] DataRobot. 2025. Time-aware data wrangling. https://shorturl.at/IdViw.
[9] Ericsson. 2025. Ericsson Mobility Report. https://shorturl.at/kG4L3.
[10] J. Breen et al. 2021. Powder: Platform for Open Wireless Data-driven Experimental Research. *Computer Networks* 197 (2021).
[11] J. Huang, D. Guo, C. Wang, J. Gu, S. Lu, J. P. Inala, C. Yan, J. Gao, N. Duan, and M. R. Lyu. 2024. Contextualized Data-Wrangling Code Generation in Computational Notebooks. In *ASE*.
[12] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *CHI*.
[13] L. Liu, S. Hasegawa, S. K. Sampat, M. Bahrami, W.-P. Chen, K. Toyota, T. Kato, T. Akazaki, A. Ura, and T. Asai. 2025. AutoDW-TS: Automated Data Wrangling for Time-Series Data. In *CIKM*.