Bahn-Vorhersage dataset: an open archive of most train delays in Germany since September 2021

Theo Döllmann

Angaben zur Veröffentlichung / Publication details:

Döllmann, Theo. 2025. “Bahn-Vorhersage dataset: an open archive of most train delays in Germany since September 2021.” Augsburg: Universität Augsburg.

Bahn-Vorhersage Dataset: An Open Archive of most Train Delays in Germany since September 2021

Theo Döllmann[1*]

[1*]University of Augsburg, Universitätsstraße 2, Augsburg, 86159, Germany.

Corresponding author(s). E-mail(s): theodor.doellmann@uni-a.de;

Abstract

Open data regarding granular train delays in Germany is scarce. This paper presents the longitudinal dataset of train delays gathered for the reliability prediction project *Bahn-Vorhersage*. Since September 2021, data has been collected via the Deutsche Bahn Timetable API, which provides both scheduled and real-time information for the entire German rail network. By implementing a polling strategy for every station, an almost complete record of train operations has been archived. The raw data is augmented with trip length information derived from OpenStreetMap and parsed into a flattened tabular format optimized for statistical analysis and machine learning applications. The dataset covers the period from September 2021 to the present and is continuously updated. Significant data gaps resulting from hardware or network outages have been documented to ensure transparency. The weekly volume of collected train stops is comparable to the official dataset published by DELFI e.V. via the National Access Point but exhibits greater temporal stability. Furthermore, punctuality statistics derived from the dataset align closely with official figures published by Deutsche Bahn, validating its representativeness. The dataset is freely available under the Open Database License (ODbL).

Keywords: Deutsche Bahn, Train Delays, Public Transport, Data Collection, Germany

1 Introduction and Motivation

The reliability of the railway network is a subject of intense public and political debate in Germany. With punctuality rates for long-distance traffic frequently dropping below 60% in recent years (Deutsche Bahn 2025a), understanding the dynamics of train delays is crucial for passengers, operators, and transport researchers alike. However, the high-quality, granular data required to analyze these dynamics is remarkably scarce. While Deutsche Bahn and other operators publish aggregate monthly statistics, detailed historical records of individual train movements are typically not preserved or made publicly available for independent analysis.

For many data science and transport projects, finding, gathering, and parsing this data represents a prohibitive barrier to entry. This challenge became apparent during the initiation of the project *Bahn-Vorhersage* (Train Prediction) in 2019, which aimed to predict the reliability of specific train connections using machine learning. This task required a substantial corpus of historic train delay data that simply did not exist in the public domain. Consequently, a custom data acquisition infrastructure was developed to record the operational state of the German rail network.

This paper documents the resulting dataset, which covers the period from September 2021 to the present. The dataset provides an almost complete record of scheduled and real-time train events (arrivals and departures) for the German rail network, including cross-border traffic. Unlike snapshot-based datasets, this archive captures the evolution of delays, allowing researchers to reconstruct how delays propagate through the network over time.

The primary contribution of this paper is the publication and documentation of this dataset to support the transport research community. The goal is to ensure that potential biases or errors in the data are transparently explained by the underlying gathering and parsing processes.

The remainder of this paper is structured as follows: Section 2 details the data source and collection methodology. Section 3 describes the data processing pipeline and schema transformations. Section 4 evaluates the dataset's characteristics and quality and section 5 analyzes common trends in the collected delay data. Finally, Section 6 summarizes the findings and discusses potential applications.

2 Data Source and Collection Methodology

Figure 1 provides a schematic overview of the data acquisition workflow described in this section. First, the interface and functionality of the primary data source are detailed, followed by the specific methodologies employed for gathering train delay data. The acquisition process entails querying all known train stops exposed by the API to retrieve both scheduled and real-time information. As the API provides distinct endpoints for static schedules and real-time updates, these data streams are harvested separately.

Over the course of the project, the collection infrastructure underwent one significant architectural revision to improve stability and coverage. This section outlines the initial setup and subsequently details the specific modifications implemented to address connection issues. For the initial description of the methodology, a static list

of EVA[1] stop identifiers is assumed. The procedure for acquiring and maintaining this list of stops is detailed in Section 2.4.

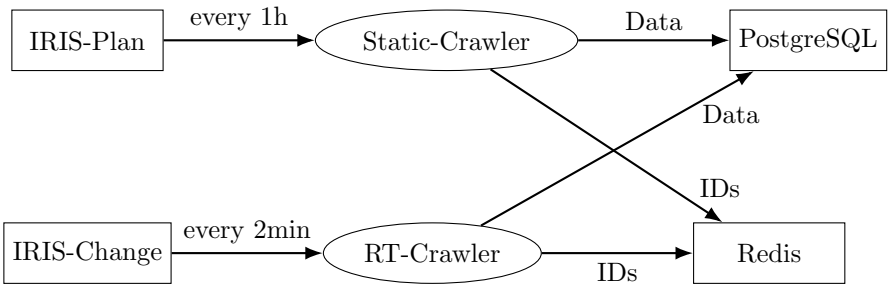


Fig. 1: Schematic overview of the data gathering process. Rectangles indicate data sources and storage systems; ellipses indicate crawling agents

2.1 Timetable API (IRIS)

The dataset presented in this study was collected via the Deutsche Bahn (DB) Timetables API (Deutsche Bahn 2025c), licensed under CC-BY 4.0. The backend system serving this API is internally known as IRIS ("Internal Passenger Information System"). In this paper, the terms IRIS and Timetable API are used interchangeably.

IRIS is an XML-based REST API comprising four primary endpoints. While official documentation is available, several critical features required for this study were undocumented and necessitated reverse engineering. Table 1 summarizes the interface. The API provides a temporal resolution of one minute[2]. The system covers both rail and rail replacement services with busses, as well as dual-system light rail vehicles (Tram-Trains, e.g., the Karlsruhe model), but excludes pure tram networks.

Listings A through C (Appendix) illustrate typical XML responses for the station *Tübingen Hbf*.

Responses from the `/plan` endpoint contain scheduled data, typically denoted by the attribute `p` (planned). Each train stop entry includes a trip label alongside arrival and/or departure events, which detail scheduled times, platform assignments, and the path (sequence of previous or subsequent stops). Every stop is assigned a unique identifier composed of three hyphen-separated components:

1. A pseudo-random numerical identifier (e.g., `6445779656564911623`), unique within a single operational day but frequently recycled for the same trip on subsequent days.
2. The date of the trip start in `YYMMDDHHmm` format (e.g., `2511181415`).
3. The stop sequence number (e.g., `3`), indicating the stop's position in the trip's itinerary.

[1] EVA numbers are unique identifiers used by Deutsche Bahn to index transport stops.
[2] Based on personal communication with DB staff, internal systems likely operate with second-level precision which is truncated for the API output. This claim could not be independently verified.

Table 1: REST endpoints of the Timetable API

Endpoint	Description
`/station/{pattern}`	Retrieves metadata for a train stop. The query pattern accepts ds100 (now ril100) codes, numeric EVA IDs, or string names. Note: While German umlauts can be URL-encoded, the character ß cannot be reliably encoded for queries. Stops containing ß must therefore be identified via alternative sources.
`/plan/{evaNo}/{date}/{hour}`	Queries the static timetable for a specific stop and hour. The hour parameter requires zero-padded digits (00-23). Date format is `YYMMDD`. Experiments indicate availability from 5 hours in the past to 12 hours in the future.
`/fchg/{evaNo}`	Retrieves all known timetable changes for a stop (*fchg* $\approx$ full changes). Updates occur every 30 seconds. Changes are purged from this list once the associated event is in the past; the exact retention period for obsolete changes is undocumented. Initial testing showed delay spikes when querying this endpoint at the top of the hour, indicating a retention period of less than one hour.
`/rchg/{evaNo}`	Retrieves changes modified within the last two minutes (*rchg* $\approx$ recent changes).

Real-time updates reference this unique ID to map changes to the static schedule. The structure of real-time events mirrors the schedule format but contains attributes prefixed with `c` (changed) for updated values. Furthermore, messages (`m`) may be attached to the stop, arrival, or departure elements. These messages typically encode delay justifications, quality-of-service alerts, or references to external systems inaccessible to the public.

2.2 Schedule Data Gathering

The acquisition of planned schedule data is not time-sensitive, as these records remain static once published. At the beginning of every hour, a thread pool is initialized, with each thread assigned a partition of EVA IDs to query. The schedule is harvested 12 hours in advance. Upon system startup, the crawler initializes by retrieving plans ranging from $t - 5$ hours to $t + 12$ hours. The precise logic is detailed in Algorithm 1.

API responses are converted from XML to JSON to facilitate downstream processing. Once a gathering cycle completes, data is upserted into a PostgreSQL database (see Section 3.1). To trigger immediate parsing of new records, the ID of each crawled datapoint is published to a Redis stream.

In August 2025, a high volume of "Connection refused" errors were observed, due to issues on the crawling server. To mitigate this, the task distribution logic was refactored. Rather than assigning fixed lists of EVAs to threads, a shared queue model was implemented. This allows threads to process EVAs dynamically, balancing the load if specific threads stall. Furthermore, an exponential backoff strategy was introduced to handle connection failures robustly.

Algorithm 1 Schedule gathering

for each $i = -5, -4, \ldots, 12$ **do**
    gather plans in i hours
end for
$h \leftarrow$ hour in 12 hours
loop
    if $h \neq$ hour in 12 hours **then**
        $h \leftarrow$ hour in 12 hours
        gather plans in h hours
    else
        wait 20 seconds
    end if
end loop

2.3 Realtime Data Gathering

Initial investigations revealed that real-time changes are purged from the API shortly after the associated event occurs. Consequently, a high-frequency polling strategy was adopted using the recent changes endpoint (`/rchg`). To minimize load on the API while ensuring data completeness, the endpoint is queried every two minutes, matching the retention window of the endpoint. This makes the operation highly time-critical.

The architecture for real-time gathering mirrors the schedule crawler. A thread pool processes a distributed list of EVA identifiers. Upon crawler startup, a full snapshot of all known changes is retrieved once; subsequently, the system polls for incremental updates. This procedure is formalized in Algorithm 2. If a collection cycle exceeds two minutes, real-time data for that interval is likely irretrievable.

Algorithm 2 Realtime gathering

gather all changes
loop
    $s \leftarrow$ current time seconds
    gather recent changes
    $w \leftarrow \max(0, 120 - (\text{current time seconds} - s))$
    wait w seconds
end loop

Data conversion and storage procedures are identical to those used for schedule data. The connection handling improvements introduced in August 2025 were simultaneously applied to the real-time crawler.

2.4 Stop Data

Comprehensive data collection requires an exhaustive index of all valid stop identifiers (EVA numbers). For the dataset to be interpretable and analytically useful, each EVA

must be associated with a human-readable name and to enable trip length calculations (see Section 3.2), geographic coordinates. The objective was to compile a station list covering the full extent of the Timetable API.

The initial seed list was derived from the DB OpenData Haltestellen dataset (DB Station&Service AG 2016). To discover previously unknown stops, two discovery methods are employed:

1. **Meta-EVAs:** Querying the `/station/` endpoint for known stops returns station objects containing `meta-evas`. These reference associated transit points (e.g., a local bus terminal referenced by the main train station).
2. **Path Extraction:** Schedule and real-time responses contain the train's full itinerary as a sequence of stop names, which can be cross-referenced against the existing database.

The stop database is updated from time to time by scanning for new `meta-evas` and stop names. Since attributes such as station names or locations can change over time, versioning is applied; if a station is renamed, all historical variants are preserved. To enrich the data, the following sources are utilized for name resolution and coordinate lookups:

- RIS::Stations API (Deutsche Bahn 2025b)
- The specific HAFAS instance used by the DB Navigator App (accessed via `hafas-client` (Redmann 2017) prior to 2025, and `db-vendo-client` (Durner 2025) thereafter)
- Community station lists from the *Travel-Status-DE-IRIS* project (Friesel 2013) (including historical and renamed station lists)
- Existing internal records
- The IRIS API (Deutsche Bahn 2025c)
- Third-party databases including Trainline (Trainline 2015) and the German Central Station Directory (ZHV) (DELFI e.V. 2019)
- Manual heuristic matching (searching station names in RIS::Stations and HAFAS to manually identify the best fit).

In cases where either the geographic location or the EVA is missing (but not both), cross-referencing is performed. If an EVA is unknown, the RIS::Stations API is queried to find nearby stops. If location data is missing, coordinates are sourced from the datasets listed above. If no match is found, the location must be determined and entered manually. As a final validation step, every potential stop is verified against the IRIS API to ensure it is active before being added to the crawling queue.

3 Data Processing and Schema

Figure 2 illustrates the data storage architecture and the transformation pipeline between formats. The primary objective of this processing pipeline is to flatten hierarchical JSON[3] data into a tabular format suitable for statistical analysis and machine learning applications. The pipeline transitions from raw data storage (Fig. 2, left)

[3]The XML responses form the API are immediately parsed to JSON

through an intermediate, GTFS-inspired (General Transit Feed Specification, see MobilityData (2025)) relational schema, finally resulting in a flat tabular representation (Fig. 2, bottom-right). We use the abbreviation *rt* to refer to real-time and use *static* to refer to schedule data, which is common GTFS practice.

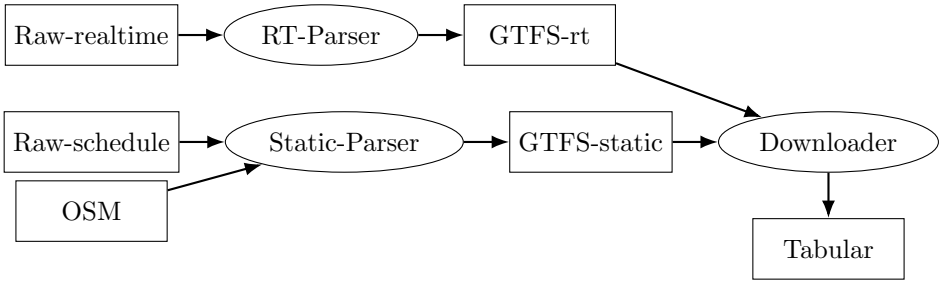


Fig. 2: Schematic of the data processing pipeline. Rectangular nodes represent storage states; elliptical nodes represent processing modules

A distinction is made between the *parsing* of raw data and the *downloading* of the GTFS-like data. The latter serves primarily as an extraction module to transfer data from the database to local storage. Raw and intermediate GTFS-like data are persisted in a PostgreSQL database, whereas the final tabular dataset is exported in the Parquet file format for efficient analytical querying.

3.1 RAW

Raw API responses are stored in two distinct PostgreSQL tables. Table 2 details the schema for schedule data, and Table 3 details the real-time counterpart. To optimize query performance, both tables are partitioned daily based on the `service_date`.

Due to historic reasons, the alphanumeric identifiers provided by the API (described in Section 2.1) are not used directly as primary keys. Instead, they are hashed into 64-bit integers. This design choice was driven by the Python Dask library, which showed significantly better performance with integer indices compared to string indices. The hashing algorithm employed is xxHash3_64 (Collet 2019). Since PostgreSQL lacks native support for unsigned 64-bit integers, a bias of 2^{63} is subtracted from the hash output to map it into the signed `BIGINT` range. This hashing strategy is applied consistently across all identifiers in the schema.

The real-time schema (Table 3) extends the schedule schema with two additional columns: `change_hash` and `time_crawled`. These fields enable the retention of multiple real-time snapshots per event, allowing for the reconstruction of delay evolution over time. Prior to September 2023, this historical evolution was not preserved, and newer updates overwrote previous states.

Table 2: Raw schedule data schema

Column	Description
service_date	Extracted date component from the ID (local time), indicating the trip start date.
hash_id	Integer hash of the unique alphanumeric identifier.
stop_id	EVA number of the stop.
plan_compressed	Full API response (JSON), Zstd-compressed and sorted by key.

Table 3: Raw real-time data schema

Column	Description
service_date	Extracted date component from the ID (local time).
hash_id	Integer hash of the unique trip ID.
change_hash	Hash of the uncompressed JSON content.
stop_id	EVA number of the stop.
time_crawled	UTC timestamp indicating when the update was captured.
change_compressed	Full API response (JSON), Zstd-compressed and sorted by key.

3.2 GTFS-static-Like Intermediate Schema

To structure the data, a relational schema inspired by the General Transit Feed Specification (GTFS) was adopted. Through iterative refinement, a stripped-down version of the standard was developed to maximize performance for this specific application. While GTFS is typically used for schedule distribution, the goal here is schedule archival. Given the potential for data gaps (e.g., missed crawls), extracting clean service patterns (e.g., calendar-based recurrences) is unfeasible, as a missing trip cannot be reliably distinguished from a non-scheduled one. Consequently, the concept of abstract *schedules* is abandoned in favor of modeling every individual trip explicitly. The data model is thus reduced to three core entities: *stop_times*, *routes*, and *stops*.

The *routes* and *stops* tables are relatively small. To reference these entities within the larger *stop_times* table, deterministic IDs are generated using the previously described hashing strategy:

- **Route ID:** Hash of the *route_long_name*.
- **Stop ID:** Hash of the concatenation of the EVA number, a specific separator, and the *platform_code*.

These IDs can be computed directly from the raw data, eliminating the need for database lookups during insertion. The *trip_id* is derived by hashing the daily unique ID and the date ID (see Section 2.1). This common identifier facilitates the later merging of static and real-time data. Like the raw tables, *stop_times* is partitioned by *service_date*.

Parsing to the GTFS-like schema is done in real-time as new raw data arrives (using the Redis stream), or as needed if the parsing was changed. Most parsing steps

are straightforward mappings from the raw data to the GTFS-like schema. However, some attributes require special handling:

- **Pickup/Drop-off Type:** The API does not explicitly define passenger boarding permissions. However, arrival or departure events may be marked as "hidden." If an event is hidden, the corresponding pickup or drop-off type is set to `NOT_ALLOWED`; otherwise, it is set to `REGULAR`. The necessity of "request-stop" interactions is not captured.
- **Betriebshalte (Service Stops):** Stops used strictly for operational purposes (where passenger exchange is prohibited) are not explicitly flagged in the source data. A stop is inferred to be a *Betriebshalt* if both arrival and departure are hidden and no platform information is provided.
- **Trip Headsign:** The headsign is not stored as a string but as the EVA number of the final stop. This is extracted from the planned path provided in the raw data. If the final stop is unknown, the last recorded stop in the path is used.
- **Shape Distance Traveled:** While exact spatial trajectories are unavailable, estimating distance is crucial for delay prediction. Distance is inferred by calculating the shortest rail path between consecutive stops using Dijkstra's algorithm on the OpenStreetMap railway network. This is not always correct but gives a good estimate. For bus services, the great-circle (haversine) distance is used as a lower bound estimate.

3.3 GTFS-rt Intermediate Schema

The approach to managing real-time updates diverges significantly from the standard GTFS-Realtime specification. The implementation avoids Protocol Buffers in favor of a flat relational structure. Only trip updates are modeled. Each update record contains:

- *Keys: service_date*, *trip_id*, and *stop_sequence* for joining with the static schedule.
- *Modifications: changed_stop_id* (indicating platform changes).
- *Delays: arrival_time* and *departure_time*.
- *Cancellations: arrival_cancelled* and *departure_cancelled*.
- *Metadata: arrival_delay_codes* and *departure_delay_codes* (delay causes and QoS messages).

These updates are stored in a partitioned PostgreSQL table. Currently, replacement services are preserved in the RAW real-time data but are not yet parsed into this intermediate structure. As with the schedule data, parsing occurs in real-time upon new data arrival or as needed.

3.4 Tabular

The final tabular format differs conceptually from the preceding schemas in one key aspect: a single data point (row) represents either an arrival or a departure event, but not both. While this increases storage requirements, it eliminates `null` values and simplifies independent analysis of arrival and departure punctuality.

Table 4 provides an in-depth explanation for this format. For mutable attributes, distinct `_schedule` and `_real` columns are maintained. It is important to note that the

`_real` columns capture the *state* of the real-time prediction at the time of crawling. A value in a `_real` column represents the actual realized time only if the boolean flag `is_final` is true.

The transformation from GTFS-static/rt to this tabular format is a straightforward join operation on *service_date*, *trip_id*, and *stop_sequence*. Auxiliary data (Stop EVAs, coordinates, route information) is joined subsequently. The most recent real-time update associated with an event is marked as final. There is no indication in the raw data whether this is actually the final update. The stop with *stop_sequence* 1 is defined as the train's origin.

For analytical purposes within the *Bahn-Vorhersage* project, trains are classified into regional and long-distance categories. This classification is based on filtering of the `category` attribute. The following categories are classified as long-distance: IC, EC, ECE, ICE, EN, RJ, RJX, TGV, FLX, NJ, THA, NEX, EIC, UEX, EST, and WB. All other categories are treated as regional services. The set of long-distance categories was curated manually.

4 Data Characteristics and Quality Assessment

4.1 Completeness and Scope

Ideally, the collection methodology outlined in Section 2 would yield a comprehensive record of all train movements. However, in practice, data gaps are introduced by network instability, hardware failures, and software defects. Quantifying absolute completeness is challenging due to the absence of a publicly available, guaranteed "ground truth" regarding the total number of scheduled rail events in Germany. Consequently, the completeness and scope of the dataset are evaluated using three proxy metrics:

1. A spatial analysis of coverage and real-time data availability.
2. A time-series analysis of collected stops and update frequencies.
3. A comparative analysis against the DELFI GTFS dataset, the official national timetable aggregate.

Figure 3 visualizes the geographic extent of the dataset. Every station with at least one recorded scheduled stop is plotted. Stations where fewer than 30% of scheduled events possess associated real-time data are highlighted in red. The dataset covers the entirety of the German rail network and extends into bordering nations and even further.

Within Germany, the majority of stations (shown in blue in Figure 3) exhibit high real-time data availability. The primary exception—stations marked in red—correlates strongly with rail replacement bus services, for which real-time telemetry is frequently unavailable via the API. See Figure D in the appendix for a comparison excluding buses. Outside German borders, real-time availability is heterogeneous.

Looking into the volume of collected data points over time in Figure 4, we must differentiate between the number of train stop events (red) and the number of collected real-time updates (blue). The red curve showing the number of train stop events remains relatively stable over the observation period, with distinct drops corresponding

Table 4: Schema of the tabular data export. All timestamps are in UTC with microsecond precision

Name	Description
trip_id	*Int64*: Unique hashed trip identifier.
time_schedule	*Datetime*: Scheduled event time.
time_real	*Datetime*: Forecasted time. Represents the actual time only if `is_final` is true.
update_timestamp	*Datetime*: UTC timestamp of when this real-time information was crawled.
delay	*Int32*: Difference in seconds between `time_real` and `time_schedule`.
dwell_time_schedule	*Int32*: Scheduled dwell time at the stop. Null if only one of arrival/departure is available.
dwell_time_real	*Int32*: Forecasted (or actual) dwell time. Null if arrival/departure is cancelled or missing.
is_final	*Boolean*: Flag indicating if this is the last known update for this event.
is_arrival	*Boolean*: True if the data point represents an arrival; False for departure.
is_cancelled	*Boolean*: Indicates if the stop was cancelled. Note: If adjacent stops are cancelled, this may be true even if the stop physically occurred.
is_regional	*Boolean*: Classification of the train as regional (True) or long-distance (False).
category	*String*: Train category (e.g., ICE, RE, RB, S).
number	*String*: Train number.
line	*String*: Line identifier (e.g., '6' for S6, 'RB61' for ag RB61 or '12' for MEX 12).
operator	*String*: Code of the railway operator.
stop_id	*Int64*: Hashed ID of the stop (links to GTFS stops).
platform_id_schedule	*Int64*: Hashed ID of the scheduled platform (links to GTFS stops).
platform_id_real	*Int64*: Hashed ID of the forecasted/actual platform (links to GTFS stops).
lat	*Float64*: Stop latitude (Assumed EPSG:4326[1]).
lon	*Float64*: Stop longitude (Assumed EPSG:4326[1]).
distance_traveled	*Int32*: Inferred distance in meters from the trip origin.
initial_scheduled_departure	*Datetime*: Scheduled departure time at the trip origin.
initial_stop_id	*Int64*: ID of the origin stop.
stop_sequence	*UInt16*: 1-based index of the stop in the trip sequence.
trip_headsign	*Int64*: ID of the destination stop.
is_betriebshalt	*Boolean*: True if the parser identified the stop as operational-only.
pickup_drop_off_type	*Enum*: `REGULAR` (0) or `NOT_ALLOWED` (1).
message_codes	*List(Int16)*: Array of message codes associated with the event.

[1]Source systems do not explicitly declare the coordinate reference system; EPSG:4326 is assumed.

to known outage events. Notably, a significant gap in January 2023 was caused by a thermal hardware failure that took the collection server offline for several weeks.

From September 2023 onwards, the collection strategy changed to retain the full history of updates rather than just the final state (indicated by the large blue area in Figure 4). This curve is inherently volatile as it depends on the actual delay evolution of trains. The real-time updates contain two major anomalies:

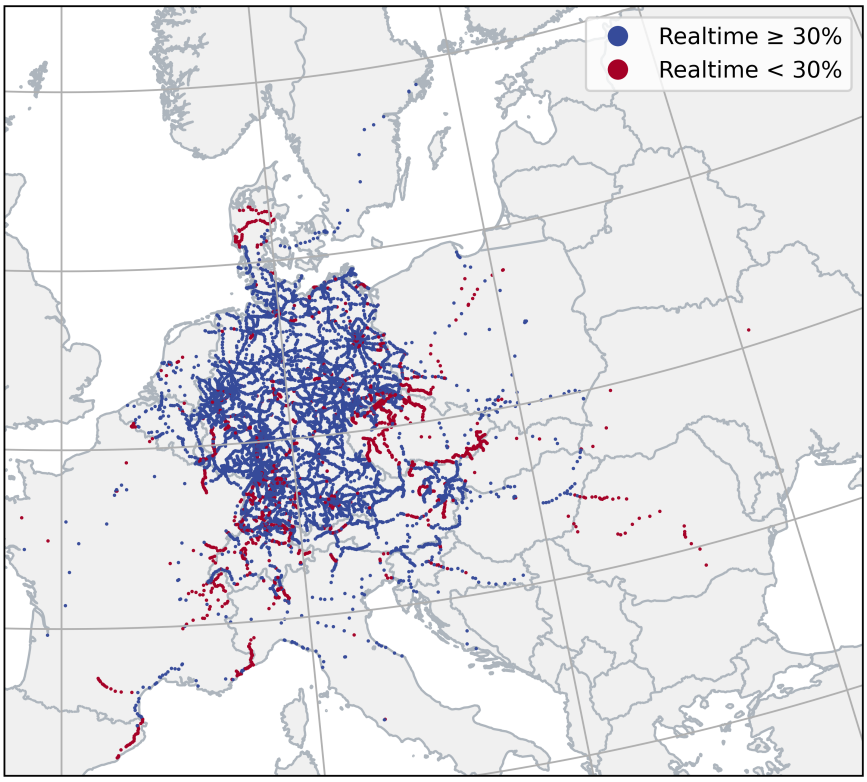


Fig. 3: Spatial coverage and real-time data availability

- **March 2024:** A gap synchronous with a drop in scheduled stops, indicating a general crawler failure.
- **September 2025:** A significant loss of real-time data while scheduled data remained stable. This was caused by a software update intended to mitigate "connection refused" errors (see Section 2.2). The update introduced a concurrency deadlock which drastically reduced throughput but did not fully halt the process, thereby failing to trigger the automated outage detection system.

The green curve in Figure 4 tracks the number of scheduled stops that received at least one real-time update. Despite the volatility in the total number of update messages, the coverage of stops remains high, except during the aforementioned September 2025 incident.

For the comparative analysis, the German GTFS dataset provided by DELFI (Durchgängige Elektronische Fahrgastinformation) is utilized. DELFI aggregates schedules from regional transport associations for the National Access Point (NAP) to fulfill EU regulations. However, the DELFI dataset is known to suffer from inconsistencies and incompleteness (Bruch 2018).

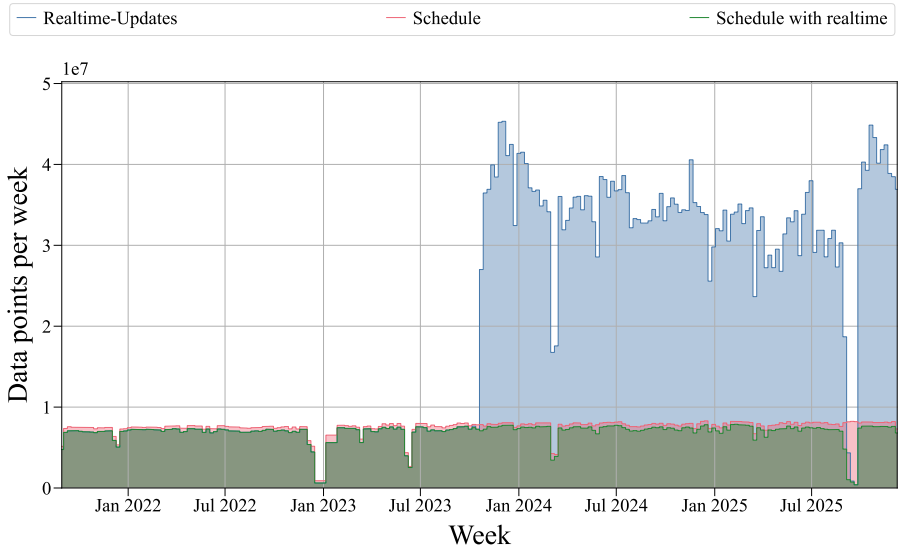


Fig. 4: Data points collected per week. Each arrival and departure event is counted as an individual data point

For this comparison, the DELFI GTFS archive (DELFI e.V. 2021a) (covering 2020–June 2025) and the current schedule (DELFI e.V. 2021b) (Dec 2025) were filtered to include only rail and rail-replacement services. For each service date, the total number of train stops was counted. This is compared against two metrics derived from our dataset:

$$N_{\text{counted}} := \sum_{\text{trips}} |\{\text{unique stop\_sequences}\}| \tag{1}$$

$$N_{\text{expected}} := \sum_{\text{trips}} \max(\text{stop\_sequence}) \tag{2}$$

N_{expected} serves as a proxy for the theoretical maximum number of stops if every scheduled stop were successfully captured.

Figure 5 presents the results. The expected count (red) consistently exceeds the actual collected count (blue), though the gap narrows significantly in 2025, likely due to better coverage of our stop dataset. The missing data found in Figure 4 is visible in this plot as well.

The DELFI reference (green) exhibits substantial volatility, fluctuating between 3.5 and 5.25 million data points per week. In contrast, our dataset shows a stable seasonal pattern. The DELFI dataset reports more stops than our actual collected count 74.3% of the time. Given that the schedule for German railways theoretically changes only twice per year (plus minor construction adjustments), the strong fluctuation of the DELFI dataset suggests internal data quality issues, such as the known misclassification of bus services as rail (Bruch 2024). Consequently, we conclude that while

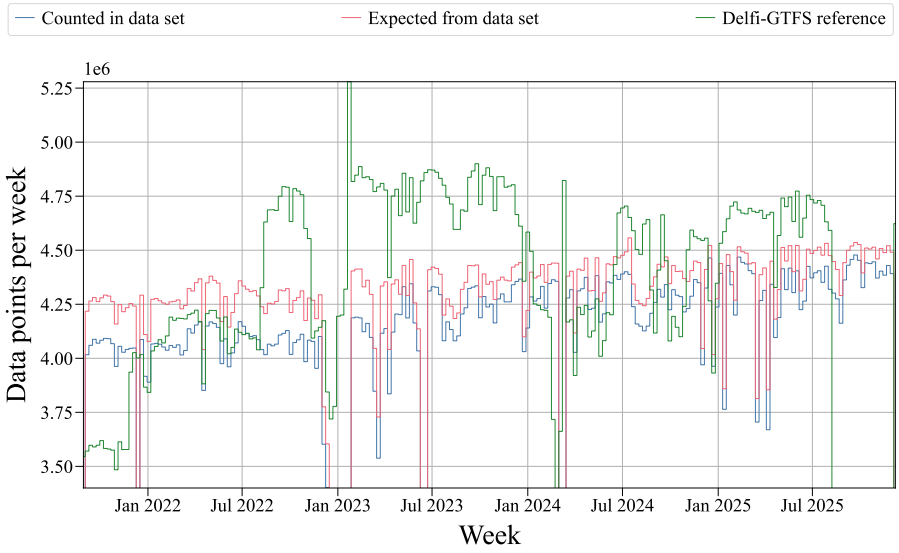


Fig. 5: Number of train stops per week in the dataset compared to the DELFI GTFS. Y-Axis zoomed to show relevant data only

our dataset is not perfectly complete, it likely offers a more consistent representation of German rail operations than the official DELFI reference.

4.2 Verification Against Deutsche Bahn Statistics

To assess the accuracy of the recorded delays, the dataset was validated against official monthly punctuality statistics published by Deutsche Bahn (Deutsche Bahn 2025a). Since official stats are deleted after one year, they were collected as part of a Git-Repository (Döllmann 2025).

Official statistics distinguish between "all traffic" and "long-distance," covering only trains operated by Deutsche Bahn. To ensure a valid comparison, our dataset was filtered using a curated list of operator codes belonging to DB (provided in Online Resource 1). Of the 40.6 million train stops in the dataset, 25.0 million (61.6%) are operated by Deutsche Bahn.

Punctuality is defined as the percentage of stops with an arrival delay of less than 6 minutes (5:59) (Deutsche Bahn 2025a). Cancelled stops are excluded from the calculation. Figure 6 compares the official figures against those derived from our data.

The alignment is strong. The average deviation for all traffic is 1.33 percentage points, and 1.50 percentage points for long-distance traffic. The largest discrepancies occur in December 2022 (2.12% deviation) and September 2025. The latter corresponds to the previously identified crawler partial outage. The December 2022 deviation also aligns with a period of missing data (see Figure 4).

Interestingly, the calculated statistics over all traffic align even more closely with official numbers when the operator filter is removed (see Figure E in the appendix). This suggests that Deutsche Bahn's internal statistics might either include third-party

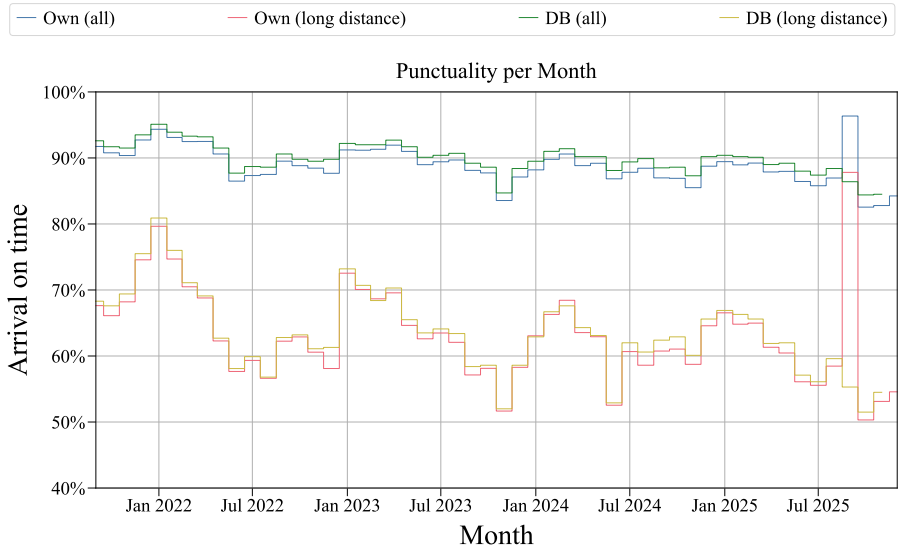


Fig. 6: Comparison between official DB punctuality statistics and those derived from the own dataset

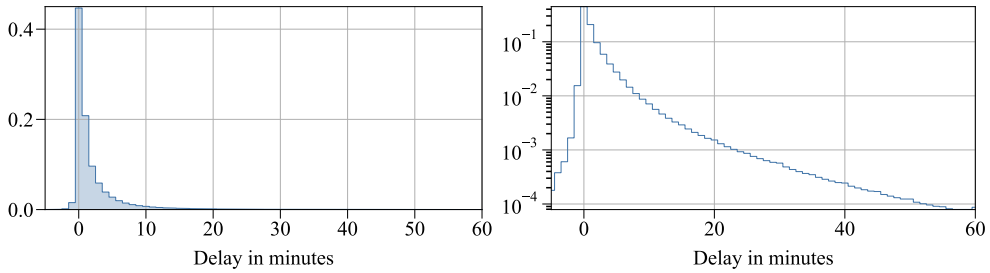
trains as part of their "all traffic" statistics, that our heuristic for identifying DB operators is overly restrictive, or it is just a coincidence. Nevertheless, the strong correlation confirms that the dataset provides a representative record of real-world punctuality. As many additional railway operators do not publish punctuality statistics, this validation is limited to Deutsche Bahn only.

5 Descriptive Statistics

This section presents a fundamental characterization of the dataset, illustrating its potential for analyzing network dynamics. The analysis focuses on three primary dimensions: the statistical distribution of delay magnitudes, temporal patterns of delay over the course of a week, and the variability of punctuality across different service classes.

Figure 7a depicts the probability density of arrival and departure delays. To improve readability, the visualization is truncated to the interval $[-5, +60]$ minutes. The distribution exhibits a pronounced heavy tail towards the right (positive delay). This characteristic is more clearly observable in the logarithmic plot (Figure 7b). Notably, the logarithmic view reveals bigger jumps at 20, 30, 40, 50, and 60 minutes. This suggests the presence of rounding artifacts.

The temporal evolution of delays is illustrated in Figure 8, which aggregates average delays into two-hour bins over a standard week. A clear pattern is evident on weekdays, characterized by two distinct peaks corresponding to the morning and evening rush hours. In contrast, weekend patterns show significantly lower average delays and the absence of a morning peak. The delay curve closely tracks network utilization



(a) Linear scale (b) Logarithmic scale

Fig. 7: Distribution of delay values across the entire dataset for both arrival and departure delays

(indicated by the gray curve of stop events), suggesting a strong correlation between network saturation and system-wide latency.

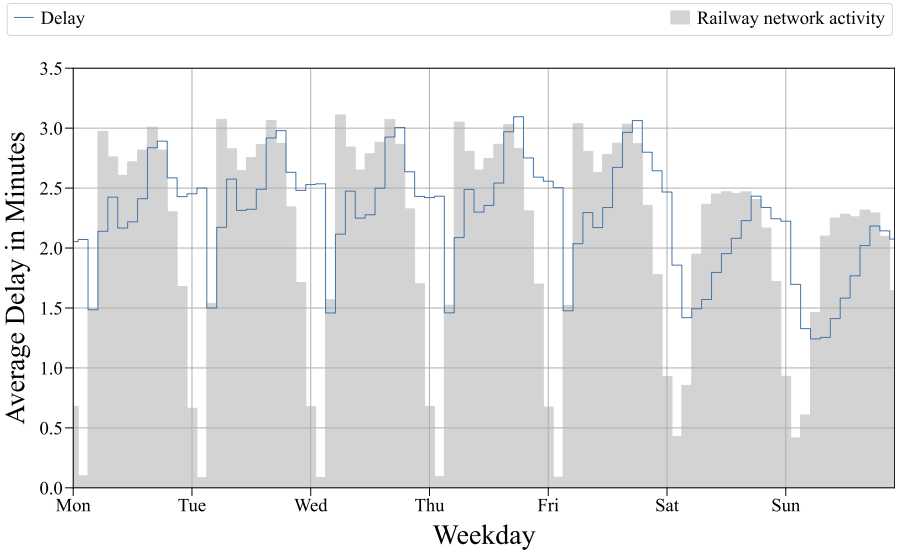


Fig. 8: Average delay over the week in two hour bins

Figure 9 shows the average delay per train category. From the size of the bubbles, it is clear that S-Bahn trains (S) make up the majority of stop events in the dataset, followed by other regional trains (RB, RE). The long-distance trains (only ICE and IC are visible) have significantly higher average delays (6 to 9 minutes) than regional trains (below 4 minutes).

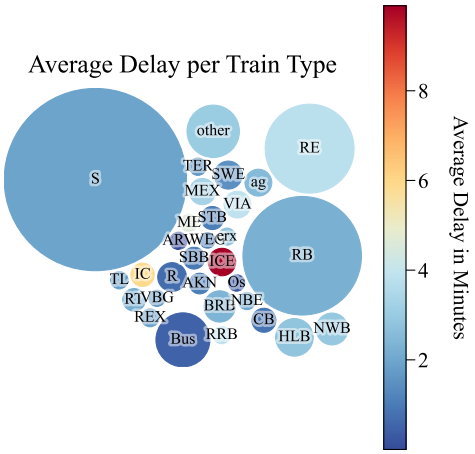


Fig. 9: Average delay vs. train category. The area of the circles corresponds to the number of stop events for that category. Categories representing less than 0.3% of all stop events are grouped into "other"

6 Conclusion and Future Work

This paper presents the *Bahn-Vorhersage* dataset, a comprehensive collection of German railway schedule and real-time data since September 2021. By continuously archiving both scheduled and real-time updates, a resource that captures the operational reality of the German rail network with high fidelity was created. The volume of collected datapoints remains stable over time, with all significant outages documented and analyzed. Validation against official Deutsche Bahn punctuality statistics confirms the dataset's representativeness and reliability for transport research. However, due to the partial data loss described in Section 4, we recommend excluding September 2025 from delay analysis.

The data gathering process is ongoing, although the operational lifespan of the Timetable API remains uncertain. Consequently, future work must focus on integrating standardized, official data sources to complement or eventually replace the current collection methodology. Potential candidates include:

- **DELFI GTFS-static & GTFS-rt**: While DELFI's GTFS data historically suffered from quality issues, recent iterations show improvement.
- **DELFI SIRI (NeTex)**: The SIRI-based real-time feed provided by DELFI appears to offer superior coverage compared to their GTFS-rt implementation.

While these official sources offer the advantage of standardized, single-request retrieval, they currently lack comprehensive real-time delay information for long-distance trains. Until official sources close this gap, the *Bahn-Vorhersage* dataset

remains a unique and essential resource for analyzing long-distance rail reliability in Germany.

7 Data Access

To ensure long-term accessibility and reproducibility, the dataset has been deposited in the German National Mobility Data Platform (*Mobilithek*), the national access point for mobility data in Germany. The data is released under the **Open Data Commons Open Database License (ODbL)**.

Due to the significant volume of the dataset, it is partitioned into three distinct repositories based on processing level. Additionally, the collected stations and parsed GTFS-Stops (referenced by the tabular dataset), are made available as well. An account on the Mobilithek platform is required to view and access the data.

- **Parsed Tabular Data** (Recommended for most researchers): https://mobilithek.info/offers/938616012299546624
- **Stations and GTFS-Stops**: https://mobilithek.info/offers/940263406913355776
- **Raw Schedule Data**: https://mobilithek.info/offers/938849406229053440
- **Raw Real-time Data**: https://mobilithek.info/offers/938859748233478144

Attribution Note: The parsed and station dataset aggregates information from multiple sources. Any reuse should attribute: *©Bahn-Vorhersage, Deutsche Bahn, OpenStreetMap contributors, Derf, Trainline, and DELFI.* For any reuse of the raw datasets, *©Bahn-Vorhersage, Deutsche Bahn* must be attributed.

7.1 Update Frequency and Live Access

The repositories on Mobilithek are updated in yearly batches. Researchers requiring higher granularity or access to the most recent data (current year) may contact the corresponding author to request access to weekly archives.

7.2 Additional Resources

Some additional resources complement the dataset:

- **Operator / Owner Codes:** A mapping of operator codes (`operator` field in the tabular dataset) to the names of the corresponding railway companies is available on GitLab: https://gitlab.com/bahnvorhersage/iris-owner-matchings.
- **Collected Deutsche Bahn Statistics**: The punctuality statistics used for validation in Section 4.2 are available on GitLab at: https://gitlab.com/bahnvorhersage/bahnvorhersage/-/blob/master/data_analysis/db_stats.py
- **Source Code**: The complete source code for data collection and processing is open-source and available on GitLab: https://gitlab.com/bahnvorhersage/bahnvorhersage.

Supplementary information. The following supplementary material is available for this article:

- **Resource_1.csv**: A comma-separated values file containing the curated list of operator codes used to filter for Deutsche Bahn-operated trains.

Appendix A Example API response: Scheduled Stop Event

Listing 1: IRIS Station

```
<station meta="752100" name="Tuebingen␣Hbf" eva="8000141" ds100="
    TT" db="true" creationts="25-11-13␣10:41:33.868"/>
```

Appendix B Example API response: Scheduled Stop Event

Listing 2: IRIS scheduled stop

```
<s id="6445779656564911623-2511181415-3">
  <tl f="N" t="p" o="800693" c="RE" n="3859" />
  <ar pt="2511181458" pp="2␣A-B" l="6" ppth="Stuttgart␣Hbf|
      Reutlingen␣Hbf" />
  <dp pt="2511181503" pp="2␣A-B" l="6" ppth="Kiebingen|Rottenburg
      (Neckar)" />
</s>
```

Appendix C Example API response: Real-time Update

Listing 3: IRIS realtime update

```
<s id="6445779656564911623-2511181415-3" eva="8000141">
  <m id="r1155906506" t="c" ts="2511181450" ts-tts="25-11-18␣
      14:50:07.429"/>
  <ar ct="2511181501" l="6">
    <m id="r34983452" t="f" c="0" ts="2511181450" ts-tts="
        25-11-18␣14:50:07.641"/>
    <m id="r34984180" t="f" c="0" ts="2511181455" ts-tts="
        25-11-18␣14:55:10.178"/>
  </ar>
  <dp ct="2511181505" l="6">
    <m id="r34983452" t="f" c="0" ts="2511181450" ts-tts="
        25-11-18␣14:50:07.641"/>
    <m id="r34984180" t="f" c="0" ts="2511181455" ts-tts="
        25-11-18␣14:55:10.178"/>
  </dp>
</s>
```

Appendix D Spatial Coverage without Buses

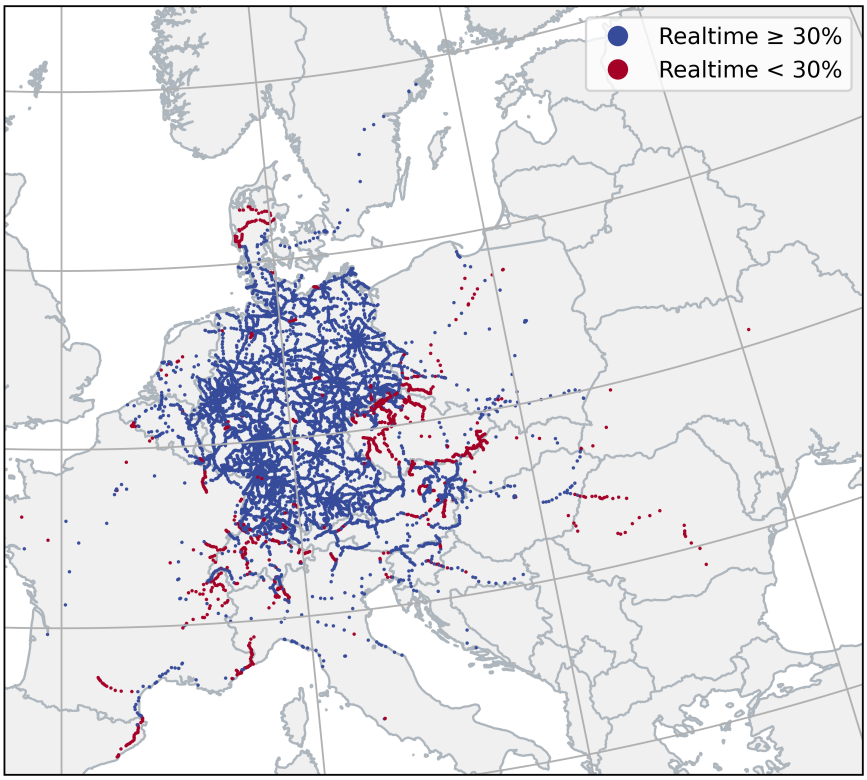


Fig. D1: Spatial coverage and real-time data availability (excluding bus services)

Appendix E DB Comparison without Operator Filter

References

Bruch, H.: GTFS-Issues. https://github.com/mfdz/GTFS-Issues Accessed 2025-12-05

Bruch, H.: DELFI (VMT, bodo,VGN): Busverkehre Als Schienenverkehre Angegeben. https://github.com/mfdz/GTFS-Issues/issues/144 Accessed 2025-12-15

Collet, Y.: xxHash. https://xxhash.com/ Accessed 2025-11-20

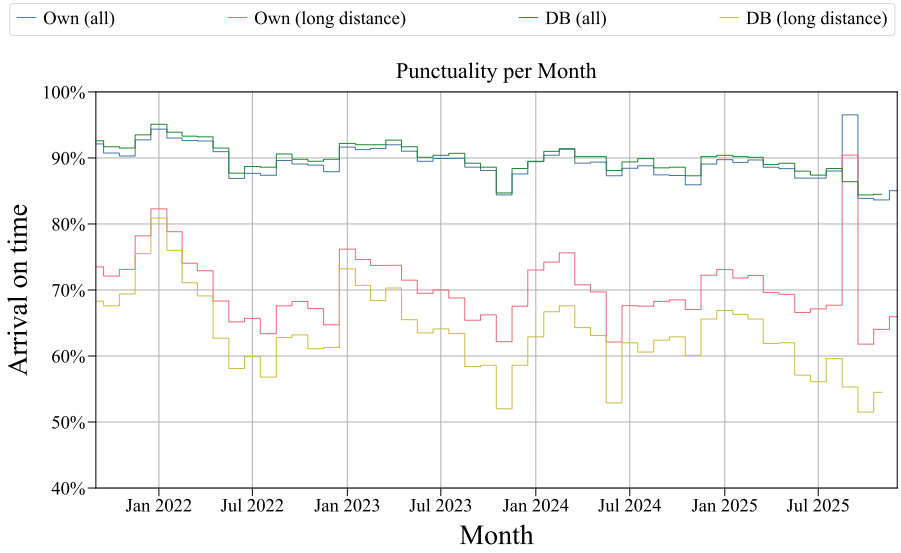


Fig. E2: Comparison between official DB punctuality statistics and those derived from the complete dataset (without operator filtering)

DB Station&Service AG: Haltestellendaten. https://web.archive.org/web/20210420012005/https://data.deutschebahn.com/dataset/data-haltestellen Accessed 2025-11-16

DELFI e.V.: Zentrales Haltestellenverzeichnis - ZHV. https://zhv.wvigmbh.de/ Accessed 2025-11-16

DELFI e.V.: DELFI-Archive. https://archiv.opendata-oepnv.de/VRR/DELFI/Soll-Fahrplandaten%20(GTFS)/ Accessed 2025-12-14

DELFI e.V.: Deutschlandweite Sollfahrplandaten (GTFS). https://www.opendata-oepnv.de/ht/de/organisation/delfi/startseite Accessed 2025-12-14

Deutsche Bahn: Deutsche Bahn: Pünktlichkeitswerte. Deutsche Bahn. https://www.deutschebahn.com/de/konzern/konzernprofil/zahlen_fakten/puenktlichkeitswerte-6878476 Accessed 2025-06-10

Deutsche Bahn: RIS::Stations. https://developers.deutschebahn.com/db-api-marketplace/apis/product/ris-stations Accessed 2025-11-16

Deutsche Bahn: Timetables – DB API Marketplace. https://developers.deutschebahn.com/db-api-marketplace/apis/product/timetables Accessed 2025-09-06

Durner, R.: Db-vendo-client. https://github.com/public-transport/db-vendo-client Accessed 2025-11-16

Döllmann, T.: Collection of Deutsch Bahn Punctuality Statistics. https://gitlab.com/bahnvorhersage/bahnvorhersage/-/blob/master/data_analysis/db_stats.py Accessed 2025-06-11

Friesel, B.: Travel-Status-DE-IRIS. https://github.com/derf/Travel-Status-DE-IRIS Accessed 2025-11-16

MobilityData: General Transit Feed Specification Reference. https://gtfs.org/ Accessed 2025-12-22

Redmann, J.: Hafas-client. https://github.com/public-transport/hafas-client Accessed 2025-11-16

Trainline: Stations - A Database of European Train Stations. https://github.com/trainline-eu/stations Accessed 2025-11-16