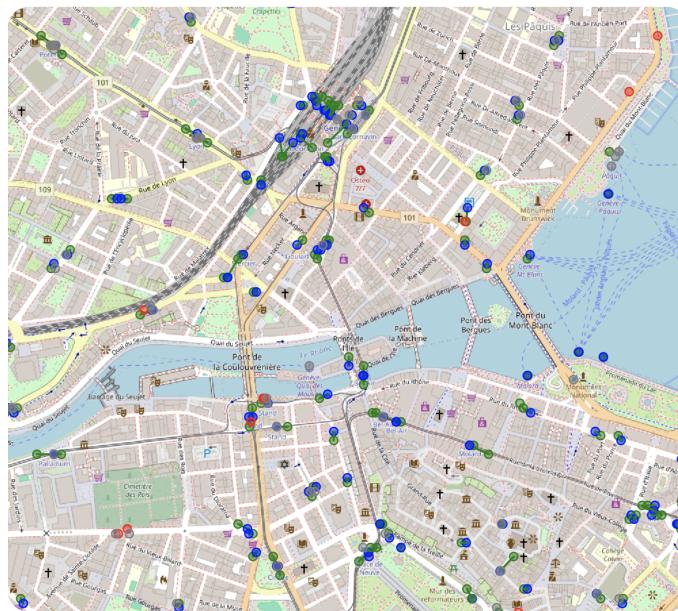


Synchronisation des Arrêts / Points d'Arrêt entre OSM et ATLAS



Travail de bachelor présenté par

Guillem MASSAGUÉ Querol

**Informatique et systèmes de communication avec orientation
Sécurité Informatique**

Août, 2025

Professeur-e HES responsable

Orestis MALASPINAS

Mandant

Matthias GÜNTER

Capture d'écran de l'application web développée dans ce projet.
Arrêts de transport public ATLAS et OSM Genève.

TABLE DES MATIÈRES

Résumé	vii
Liste des illustrations	xii
Liste des tableaux	xiii
Introduction	1
1 Chapitre 1 : Données ATLAS	4
1.1 Arrêts	4
1.2 GTFS	7
a Description des fichiers	7
b Clés d'identification, normalisation et jointure	8
c Résultats	10
1.3 HRDF	10
a Comment nous le lisons	11
b Informations exploitées	12
c Statistiques clés	12
1.4 Comparaison GTFS et HRDF (couverture SLOID)	15
2 Chapitre 2 : Transport public dans OSM	17
2.1 Schémas de cartographie du transport public dans OSM	17
2.2 Différences dans l'usage de clés spécifiques	18
2.3 Requête Overpass transport public — Suisse	19
2.4 Aperçu des balises (<i>tags</i>) des nœuds OSM en Suisse	20
2.5 Aperçu des itinéraires de transport public dans OSM en Suisse	22
a Les 5 nœuds de transport public les plus « connectés »	23
b Analyse des directions des itinéraires	24
c Top 5 des itinéraires avec le plus d'arrêts	24
3 Chapitre 3 : Correspondance avec les données ATLAS-OSM	26
3.1 Approche méthodologique générale	26
3.2 Correspondance Exacte	27
3.3 Correspondance par Nom	27
3.4 Correspondance par Distance	27
a Étape 1 : Correspondance de groupe basée sur la proximité	28
b Étape 2 : Correspondance par référence locale dans un rayon de 50 mètres	32
c Étape 3 : Correspondance basée sur la proximité avec critères relatifs	32
3.5 Consolidation Post-traitement	33
3.6 Propagation des Duplicatas	34
3.7 Correspondance Manuelle	34
3.8 Résultats actuels	34
4 Chapitre 4 : Lignes et appariement par lignes	36
4.1 Des flux bruts à un fichier unique de lignes	36

a	Ce qu'est <code>atlas_routes_unified.csv</code>	36
b	Comment on le génère (vue d'ensemble)	37
4.2	Ce que disent les données unifiées	37
4.3	D'autres vues utiles	40
4.4	Fonctionnement de l'appariement par lignes	42
a	Candidats par distance	42
b	Paramètres	43
4.5	Résultats de l'appariement par lignes	43
a	Efficacité de l'appariement par lignes : une analyse complète	44
4.6	Bilan et perspectives	45
5	Chapitre 5 : Analyse des résultats	46
5.1	Lecture globale par méthode	46
5.2	Par opérateur : où est-ce le plus précis ?	48
5.3	Répartition par tranches de distance	48
5.4	Cas extrêmes : top 10 plus grandes distances	48
6	Chapitre 6 : Détection des problèmes	50
6.1	Types de problèmes détectés	50
6.2	Nécessité de la priorisation	50
6.3	Méthode d'attribution des priorités	50
a	Priorisation des problèmes de distance	50
b	Priorisation des non-appariements	51
c	Priorisation des problèmes d'attributs	51
6.4	Interface de l'application web	52
6.5	Statistiques	52
6.6	Persistance et flux de travail de révision	54
7	Chapitre 7 : Base de données et données persistantes	55
7.1	Importer les données : le rôle de <code>import_data_db.py</code>	55
7.2	Schéma logique : les tables qui comptent	56
7.3	Pros et cons du schéma vis-à-vis du rendu cartographique	58
7.4	Données persistantes : comment elles survivent aux ré-imports	59
7.5	Performance : pourquoi ça défile vite sur la carte	60
8	Chapitre 8 : Backend	61
8.1	Architecture générale	61
8.2	Blueprints et endpoints	62
a	Données : <code>/api/data</code>	62
b	Recherche et matches manuels	62
c	Statistiques : <code>/api/global_stats</code>	62
d	Rapports : <code>/api/generate_report</code>	63
e	Problèmes et persistance	63
8.3	Requêtage partagé et optimisation	63
8.4	Diagrammes de flux	64
8.5	Bonnes pratiques et lisibilité	64
9	Chapitre 9 : Frontend	66

9.1	Pages et navigation	66
9.2	Carte interactive (Index)	67
a	Cycle de vie des requêtes	67
b	Rendu des marqueurs et des lignes	68
c	Info-bulles et chargement paresseux	68
d	Panneau de filtres et recherche	68
9.3	Outil Problèmes	69
9.4	Données persistantes	70
9.5	Rapports et instantanés	70
9.6	Liaison avec le backend (rappel Chap. 8)	71
10	Chapitre 10 : Système d'authentification sécurisé	74
10.1	Objectifs	74
10.2	Aperçu de l'architecture	74
a	Composants	74
10.3	Modèle de données (auth_db)	75
a	Parcours utilisateur et impact sur les données	76
10.4	Sécurité opérationnelle (en pratique)	80
11	Chapitre 11 : Évaluation de la sécurité de l'application	82
11.1	Périmètre et surface d'attaque	82
11.2	Contrôles en place	83
11.3	Scénarios d'attaque et déroulé	87
11.4	Calibrage et limites actuelles	89
11.5	Conclusion	89
12	Chapitre 12 : Déploiement et Conteneurisation	90
12.1	Introduction à Docker	90
12.2	Les quatre pièces du puzzle	90
12.3	L'orchestration : docker-compose.yml	91
12.4	L'image : dockerfile	93
12.5	Le chef d'orchestre : entrypoint.sh	94
12.6	Démarrer en 30 secondes	95
12.7	Variables d'environnement	96
12.8	Persistante et poids d'image	96
12.9	Considérations de sécurité	97
Conclusion	99	
Références documentaires	103	

À ma famille, mes amis... et aux trains qui arrivent à l'heure.

RÉSUMÉ

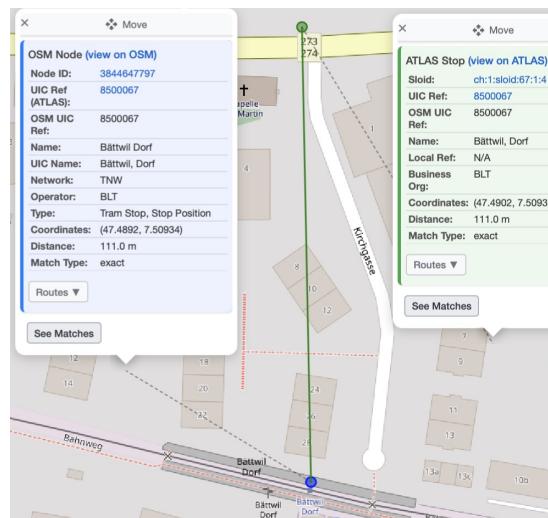
Ce projet vise à synchroniser les arrêts de transport public présents dans OpenStreetMap (OSM) avec ceux du système officiel suisse ATLAS, afin d'améliorer la précision et la fiabilité des données. Nous analysons d'abord la structure, la couverture et les balises des deux sources (ATLAS/OSM) en Suisse, en mobilisant les jeux GTFS et HRDF pour caractériser les lignes et leurs directions.

Nous mettons ensuite en place un pipeline d'appariement séquentiel combinant plusieurs méthodes (exacte, par nom, par distance, par lignes GTFS/HRDF), suivi d'une consolidation et de la gestion des doublons. Parmi **55 571** arrêts ATLAS, nous établissons **48 419** correspondances, soit **84,3%** d'arrêts ATLAS distincts appariés. Répartition par méthode : **21 237** exactes, **18 618** par distance, **7 021** par lignes, **537** par nom.

Un système de **détection et de priorisation des problèmes** met en évidence **16 408** anomalies de priorité **P1** (distance : **1 171**; non-appariements : **8 192**; attributs : **7 045**), guidant la revue ciblée.

Nous développons une application web permettant de visualiser les deux ensembles et leurs correspondances, de produire des rapports, de corriger les problèmes détectés et d'effectuer des correspondances manuelles avec **persistence** des décisions.

Enfin, l'application intègre un volet de sécurité aligné sur les bonnes pratiques (mots de passe Argon2, 2FA TOTP, CSRF, limitation de débit) et une conteneurisation Docker assurant un déploiement reproductible. L'ensemble constitue une base pour améliorer durablement la qualité des données et accélérer la synchronisation OSM–ATLAS.



Exemple de problème de priorité 1 : Bättwil, Dorf

Candidat-e :

GUILLEM MASSAGUÉ QUEROL

Filière d'études : ISC

Professeur-e(s) responsable(s) :

ORESTIS MALASPINAS

En collaboration avec : SKI+

Travail de bachelor soumis à une convention de stage
en entreprise : non

Travail soumis à un contrat de confidentialité : non

ACRONYMES

- 2FA** Two-Factor Authentication — authentification à deux facteurs.
- API** Application Programming Interface — interface de programmation applicative.
- ATLAS** Référentiel officiel suisse des arrêts (plateforme Open Transport Data Switzerland).
- AWS** Amazon Web Services — plateforme de services cloud.
- BBox** Bounding Box — rectangle englobant géographique.
- CAPTCHA** Completely Automated Public Turing test to tell Computers and Humans Apart — test anti-robot.
- CDN** Content Delivery Network — réseau de diffusion de contenu.
- CFF** Chemins de fer fédéraux suisses (voir aussi SBB).
- CSP** Content Security Policy — politique de sécurité de contenu.
- CSRF** Cross-Site Request Forgery — falsification de requête inter-sites.
- CSV** Comma-Separated Values — format de fichier tabulaire.
- DOM** Document Object Model — modèle objet du document (navigateur).
- DoS** Denial of Service — déni de service.
- GTFS** General Transit Feed Specification — format d'échange pour horaires et arrêts.
- HRDF** HAFAS Raw Data Format — format brut d'horaires (transport ferroviaire).
- HTTP** HyperText Transfer Protocol — protocole de transfert hypertexte.
- HTTPS** HyperText Transfer Protocol Secure — HTTP chiffré via TLS.
- IP** Internet Protocol — protocole Internet (adresse IP).
- JSON** JavaScript Object Notation — format d'échange de données.
- KD-tree** k-dimensional tree — structure de données pour recherches de voisinage.
- LRU** Least Recently Used — politique de cache « moins récemment utilisé ».
- LV95** Système de coordonnées suisse CH1903+ / LV95.
- ORM** Object–Relational Mapping — mappage objet-relationnel.

OSM OpenStreetMap — projet cartographique collaboratif.

PDF Portable Document Format — format de document portable.

PTv1 Public Transport version 1 — schéma OSM historique pour le transport public.

PTv2 Public Transport version 2 — schéma OSM moderne pour le transport public.

QR Quick Response — code à réponse rapide (QR code).

RCE Remote Code Execution — exécution de code à distance.

SARGable Search ARGument able — requêtes exploitant directement des index SQL.

SBB Schweizerische Bundesbahnen — Chemins de fer fédéraux suisses (voir aussi CFF).

SDK Software Development Kit — kit de développement logiciel.

SES (Amazon) Simple Email Service — service d'envoi d'emails.

SLOID Identifiant d'arrêt ATLAS (identifiant interne des plateformes/quais).

SQL Structured Query Language — langage de requêtes relationnelles.

SVG Scalable Vector Graphics — format vectoriel pour le web.

TOTP Time-based One-Time Password — mot de passe à usage unique basé sur le temps.

UIC Union Internationale des Chemins de Fer — organisation internationale des chemins de fer (codes pays/gares).

UID/GID User ID / Group ID — identifiants utilisateur et groupe sous Unix.

UI User Interface — interface utilisateur.

URL Uniform Resource Locator — localisateur uniforme de ressource.

WGS84 World Geodetic System 1984 — système géodésique mondial (coordonnées GPS).

XML eXtensible Markup Language — langage de balisage extensible.

LISTE DES ILLUSTRATIONS

1.1	ATLAS : distribution nationale	6
1.2	Désignations et opérateurs ATLAS	6
1.3	GTFS : lignes par SLOID	11
1.4	Arrêts GTFS (Suisse)	12
1.5	Quais HRDF – Suisse	14
1.6	ATLAS – Genève	15
1.7	Genève : GTFS vs HRDF (points bruts)	15
1.8	Genève : SLOIDs appariés (GTFS vs HRDF)	16
2.1	Présence des balises clés	22
2.2	Distribution des itinéraires par nœud	23
2.3	Directions H/R connues	24
3.1	Correspondances exactes à Genève-Cornavin	28
3.2	Exemple de correspondance par nom	29
3.3	Correspondances – Münchenstein, Hofmatt	30
3.4	Correspondances distance étape 2	32
3.5	Correspondance par distance – étape 3a	33
3.6	Correspondance par distance – étape 3b	34
4.1	Nœuds OSM hors vs. sur lignes (Genève)	38
4.2	OSM : tracé des lignes (Genève)	38
4.3	Atlas-GTFS : approximation des trajets (Genève)	39
4.4	Distribution du nombre de lignes GTFS uniques par <code>sloid</code> (coupée à 20).	40
4.5	Distribution du nombre de couples (ligne, direction) par <code>sloid</code> (coupée à 30).	40
4.6	Distribution du nombre de lignes HRDF uniques par <code>sloid</code> (coupee a 20).	41
4.7	Nombre de tokens GTFS (<i>ligne,direction</i>) par <code>sloid</code> retrouvés dans OSM (coupée à 30).	45
5.1	Boîtes par méthode	46
5.2	Histogrammes par méthode (0–50 m)	47

5.3	Distances par opérateur	48
5.4	Tranches de distance	48
6.1	Problèmes par type	53
6.2	Priorités par type	53
6.3	Opérateurs — part des P1	54
7.1	Schéma relationnel des tables	57
8.1	Flux /api/data	64
8.2	Cycle de persistance	64
9.1	Page d'accueil	66
9.2	Filtre ATLAS (Zurich)	67
9.3	Bannière de zoom	68
9.4	Info-bulles	69
9.5	Filtres — chips actifs	70
9.6	Résumé d'en-tête	70
9.7	Page Problèmes — vue d'ensemble	71
9.8	Contrôles — Page Problèmes	72
9.9	Panneau de résolution	72
9.10	Données persistantes (non admin)	73
9.11	Page Rapports	73
10.1	Bouton Créer un compte	76
10.2	Inscription	76
10.3	users après inscription	77
10.4	Message vérification email	77
10.5	users — champs statut	78
10.6	Page de connexion	78
10.7	Connexion réussie	78
10.8	users — verrous	79
10.9	Activation 2FA	79
10.10	Codes de secours	80

10.11 Saisie 2FA	80
10.12 users — champs 2FA	80
12.1 Architecture Docker (app+db)	91
12.2 Commande compose up	95
12.3 Vue Docker Desktop	96
12.4 Distribution du code Python : nombre de fichiers par catégorie (gauche) et lignes de code par catégorie (droite)	101

LISTE DES TABLEAUX

1.1	Extrait du fichier <code>stops.txt</code>	7
1.2	Extrait du fichier <code>routes.txt</code>	8
1.3	Extrait du fichier <code>trips.txt</code>	8
1.4	Extrait du fichier <code>stop_times.txt</code>	8
1.5	Extrait de <code>stops.txt</code> pour "Lancy-Pont-Rouge"	9
1.6	Extrait de <code>traffic-points-actual-data</code> pour "Lancy-Pont-Rouge"	9
1.7	Extrait de <code>stops.txt</code> pour "Lausanne Bourdonnette"	10
1.8	Extrait de <code>traffic-points-actual-data</code> pour "Lausanne Bourdonnette"	10
3.1	Données ATLAS – Münchenstein, Hofmatt	31
3.2	Données OSM – Münchenstein, Hofmatt	31
4.1	Structure du fichier <code>atlas_routes_unified.csv</code>	36
5.1	Top 10 distances	49
6.1	Indicateurs clés de la dernière exécution	52
6.2	Répartition des problèmes par type et priorité	52
6.3	Répartition détaillée des non-appariements	52
10.1	Champs de la table <code>users</code>	75
10.2	Champs de la table <code>auth_events</code>	75
12.1	Variables d'environnement pour la personnalisation	96
12.2	Correspondances par méthode (sur 48 419 correspondances)	99
12.3	Problèmes de priorité P1 par type	100
12.4	Résumé des lignes de code par type de fichier	100

INTRODUCTION

CONTEXTE ET PROBLÉMATIQUE

La numérisation des services de mobilité rend la qualité des données de transport public plus cruciale que jamais. En Suisse, les systèmes d'information voyageurs, les planificateurs d'itinéraires et les outils d'analyse reposent sur des référentiels géographiques précis des arrêts. ATLAS constitue la base officielle, tandis qu'OpenStreetMap (OSM), projet cartographique collaboratif mondial, offre une richesse et une couverture exceptionnelles.

Au cœur de ce travail se trouvent leurs **divergences**. Deux sources décrivent la même réalité — le réseau et ses arrêts — mais avec des identifiants, des hiérarchies et des coordonnées non synchronisés. Ces décalages, parfois de quelques mètres, parfois substantiels, créent des incohérences qui affectent :

- **Les usagers** : informations contradictoires, localisation imprécise, expérience utilisateur dégradée
- **Les exploitants et planificateurs** : difficultés à optimiser les lignes, les correspondances et l'infrastructure

Nous proposons une approche systématique visant à **identifier, apparié et corriger** les divergences entre ATLAS et OSM, puis à **faciliter la décision humaine** lorsque l'automatisation atteint ses limites.

Objectifs

- Concevoir une méthodologie robuste d'appariement ATLAS ↔ OSM (identifiant, nom, distance, lignes)
- Quantifier les incohérences (positions, attributs, non-appariements)
- Développer une application web de visualisation, de triage et de correction avec *persistence* des décisions
- Générer des rapports et prioriser les problèmes
- Mettre en place un système d'authentification sécurisé et évaluer la robustesse de l'application
- Fournir un déploiement reproductible (conteneurs) et un code ouvert

APPROCHE MÉTHODOLOGIQUE

Notre démarche s'articule en trois phases principales, allant du traitement brut des données à la validation humaine appuyée par un outil sécurisé.

Phase 1 : Pipeline de traitement automatisé

Le cœur de notre approche est un pipeline de scripts Python qui exécute les tâches suivantes :

- **Collecte et Prétraitement** : Importation des données brutes depuis ATLAS (CSV) et OpenStreetMap (API Overpass), suivie d'un nettoyage et d'une normalisation
- **Appariement Algorithmique** : Application d'une cascade de méthodes : appariement par identifiants, par noms, par proximité géographique (via un index spatial R-tree) et enfin par analyse des lignes de transport
- **Détection des Problèmes** : Identification automatique des arrêts non appariés, des doublons potentiels et des incohérences, ensuite classés et priorisés pour l'étape suivante

Phase 2 : Application web pour la visualisation et la validation humaine

Les cas problématiques sont présentés dans une application web interactive, conçue pour faciliter et accélérer la prise de décision ainsi que la correction des données :

- **Architecture Technique** : Une application full-stack avec un backend Python (Flask) exposant une API, une base de données MySQL, et un frontend en JavaScript natif avec une interface cartographique (Leaflet)
- **Interface Utilisateur** : Une interface efficace pour visualiser les appariements et les problèmes, trier par priorité et appliquer des solutions manuelles
- **Persistance des Corrections** : Chaque correction est conservée de façon durable, puis réinjectée au pipeline lors des futures mises à jour des données, créant une boucle d'amélioration continue

Phase 3 : Sécurisation et Déploiement

La finalité du projet est de fournir un outil robuste et utilisable en conditions réelles :

- **Sécurité** : Mise en place d'un système d'authentification complet (mots de passe hachés, authentification à deux facteurs) et évaluation des vulnérabilités pour protéger les données et les actions des utilisateurs
- **Déploiement** : L'ensemble de l'application et de ses dépendances (base de données, backend, frontend) est conteneurisé avec Docker, permettant une installation et une mise en service simplifiées et reproductibles

L'intégralité du code est disponible sur :

GitHub : https://github.com/openTdataCH/stop_sync_osm_atlas

GitHEPIA : <https://githepia.hesge.ch/guillem.massague/bachelor-project>

Ce document ne présente que des extraits ciblés.

Conventions de couleur

Dans les captures d'écran de l'application web :

- **Points verts** : arrêts ATLAS avec correspondance OSM
- **Points bleus** : nœuds OSM correspondants
- **Points rouges** : arrêts ATLAS sans correspondance
- **Points gris** : nœuds OSM sans correspondance

STRUCTURE DU MÉMOIRE

Le mémoire suit un cheminement logique, partant des données brutes pour aboutir à une solution logicielle complète et déployable.

Nous commençons par poser les fondations en présentant les deux univers de données au cœur de ce projet : les données officielles suisses avec **ATLAS (Chapitre 1)**, puis leur contre-partie collaborative mondiale, **OpenStreetMap (Chapitre 2)**.

Une fois ces deux mondes définis, nous nous attelons à les réconcilier. Le **Chapitre 3** détaille les **méthodes d'appariement** par identifiant, nom et proximité géographique. Le **Chapitre 4** introduit une approche plus fine, exploitant les **lignes de transport** pour affiner les correspondances.

L'**analyse des résultats (Chapitre 5)** quantifie ensuite le succès de notre approche automatisée, en examinant la qualité des appariements. Ce qui nous conduit naturellement à la **détection systématique des problèmes (Chapitre 6)**, où nous classons et priorisons les cas restants qui nécessitent une intervention humaine.

Pour faciliter cette intervention, nous développons une application web. Sa conception est détaillée en trois temps : d'abord l'architecture de la **base de données (Chapitre 7)**; ensuite, la logique métier du **backend (Chapitre 8)**; et enfin, l'interface utilisateur interactive du **frontend (Chapitre 9)**, illustrée par de nombreuses captures.

Puisque la robustesse d'un tel outil repose sur sa sécurité, nous consacrons deux chapitres à cet aspect crucial. Le **Chapitre 10** décrit la mise en place d'un **système d'authentification** moderne et complet, tandis que le **Chapitre 11 évalue la sécurité** globale de l'application. Enfin, le **Chapitre 12** assure la reproductibilité et la simplicité du **déploiement** grâce à la conteneurisation, permettant de mettre en service l'ensemble de la solution en une seule commande.

Une **conclusion** synthétise les apports de ce travail, ses limites et les pistes pour de futures améliorations.

CHAPITRE 1 : DONNÉES ATLAS

Les données ATLAS, au cœur de cette étude, proviennent de la plateforme suisse Open Transport Data¹. Cette ressource centralise les données des transports publics en Suisse, offrant une base précieuse pour l'analyse et le développement d'applications.

Reproductibilité. Les figures et statistiques de ce chapitre ont été produites par des scripts reproductibles disponibles dans le répertoire `memoire/scripts_used/` du dépôt Git du projet.

1.1. ARRÊTS

L'analyse des arrêts s'appuie sur le jeu de données `traffic-points-actual-data`², qui recense les arrêts de transport public en Suisse en fournissant des informations détaillées sur leur localisation et leurs caractéristiques. Ces points peuvent être visualisés sur une carte interactive via l'application web ATLAS³.

Nous nous concentrons ici sur deux colonnes principales : le `number` et la `designation` de chaque arrêt.

Le numéro d'un arrêt correspond à la référence UIC (Union Internationale des Chemins de fer), un standard international pour l'identification des lieux de transport public. Les deux premiers chiffres représentent le code du pays ; la Suisse, par exemple, utilise le code 85⁴. Ainsi, un numéro UIC comme 8502034 désigne un arrêt spécifique du réseau suisse.

La colonne `designation` fait référence à une identification locale : une valeur de 3 peut, par exemple, indiquer que l'arrêt correspond à la plateforme 3 d'une gare.

Enfin, les données incluent également des informations sur l'opérateur responsable de chaque arrêt, un élément utile pour établir des correspondances avec d'autres jeux de données.

Le jeu de données distingue deux types de `trafficPointElementType` : `BOARDING_AREA` et `BOARDING_PLATFORM`. Notre analyse se limite aux `BOARDING_PLATFORM`, car les `BOARDING_AREA` ne disposent pas de coordonnées géographiques. Pour extraire ces informations, nous avons développé un script Python, `get_atlas_data.py`. Extrait simplifié du chargement/filtrage :

-
1. Open Data Platform Mobility Switzerland [en ligne]. Disponible sur : <https://opentransportdata.swiss/en/> (consulté le 2025-07-21).
 2. Traffic-points-actual-date [en ligne]. Disponible sur : <https://data.opentransportdata.swiss/en/dataset/traffic-points-actual-date> (consulté le 2025-08-01).
 3. ATLAS App SBB [en ligne]. Disponible sur : <https://atlas.app.sbb.ch> (consulté le 2025-07-23).
 4. Wikipédia. Liste des codes pays UIC [en ligne]. Disponible sur : https://fr.wikipedia.org/wiki/Lista_des_codes_pays_UIC (consulté le 2025-08-02).

Extrait — get_atlas_stops

```

1 def get_atlas_stops(output_path, download_url):
2     response = requests.get(download_url)
3     response.raise_for_status()
4     with zipfile.ZipFile(io.BytesIO(response.content)) as z:
5         csv_filename = z.namelist()[0]
6         with z.open(csv_filename) as f:
7             df = pd.read_csv(f, sep=';')
8             # Suisse (UIC pays = 85)
9             df = df[df['uicCountryCode'] == 85]
10            # Filtre frontière CH: inclusion dans le polygone suisse (WGS84)
11            df = filter_points_in_switzerland(df, lat_col='wgs84North', lon_col='
12 wgs84East')
13            # Comptage des quais
14            boarding_platforms = df[df['trafficPointElementType'] == 'BOARDING_PLATFORM']
15            df.to_csv(output_path, sep=';', index=False)

```

Statistiques ATLAS. Sur l'instantané analysé (après filtre UIC=85 et frontière CH par polygone) :

- **Lignes avec coordonnées (dans la frontière CH)** : 55 571.
- **BOARDING_PLATFORM** : 54 882.
- **UIC distincts (number)** : 26 750.
- **designation non vides** : 12 051 (538 valeurs distinctes).
- **designation manquantes** : 43 520, dont 4 365 cas où l'unique entrée du number est sans désignation.
- **Entrées identifiables par (number, designation) seul** : 10 514.
- **Total identifiables par number + (designation ou unicité du number)** : 15 854.

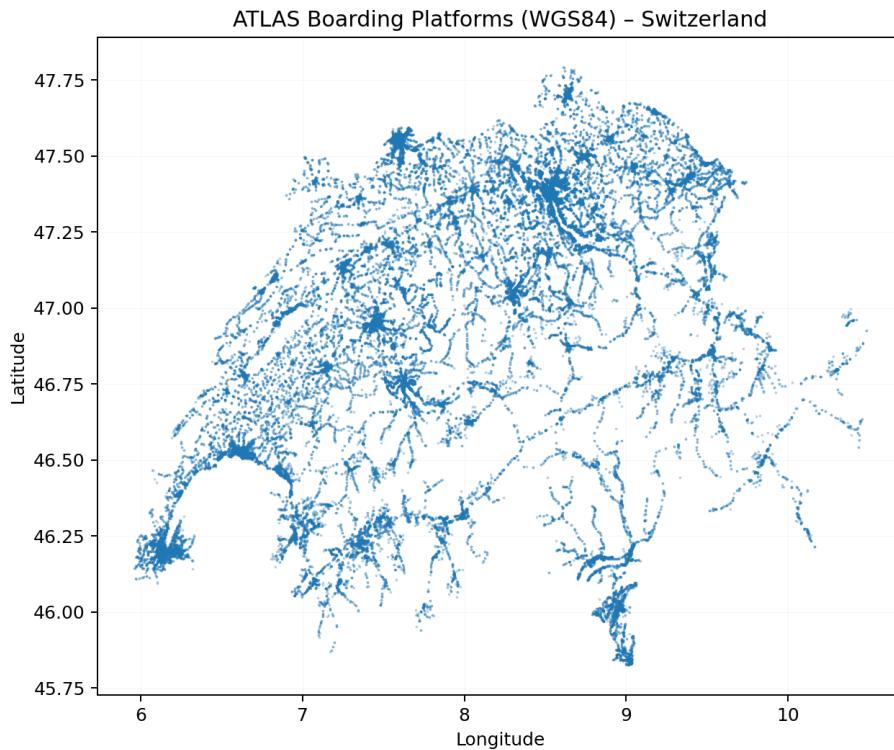


ILLUSTRATION 1.1 – ATLAS : distribution nationale (WGS84).

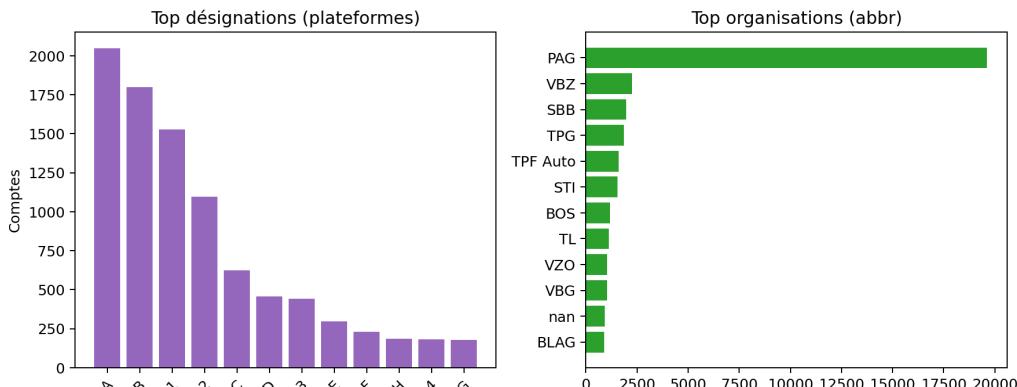


ILLUSTRATION 1.2 – À gauche : désignations de plateforme les plus fréquentes (hors valeurs manquantes). À droite : principales organisations (abrév.) déclarées.

Remarque. Les très nombreuses valeurs manquantes pour designation sont exclues du classement afin d'éviter un effet disproportionné. Comme indiqué plus haut, 43 520 entrées n'ont pas de designation.

Concernant les coordonnées, le fichier fournit deux systèmes : le système de référence suisse LV95 et le système de référence global WGS84. Étant donné que les données d'OpenStreetMap (OSM) utilisent les coordonnées WGS84, nous nous concentrerons sur cet ensemble de coordonnées.

1.2. GTFS

Le General Transit Feed Specification (GTFS) est un format d'échange numérique initié par Google pour standardiser les horaires des transports publics et leurs informations géographiques, telles que la localisation des arrêts. En Suisse, ces données sont publiées sur la plateforme OpenTransportDataSuisse³⁰. Elles servent à développer des applications pratiques, comme les outils de consultation d'horaires ou de planification de trajets.

Bien que notre projet se focalise actuellement sur la synchronisation des arrêts, les données GTFS relatives aux trajets retiennent également notre attention. Elles pourraient faciliter la correspondance entre les arrêts ATLAS et ceux d'OpenStreetMap, en exploitant les informations sur les itinéraires présentes dans les deux ensembles de données. Parmi les fichiers GTFS, quatre retiennent notre attention : `stops.txt`, `stop_times.txt`, `routes.txt` et `trips.txt`.

a. Description des fichiers

Les fichiers GTFS suivants sont cruciaux pour notre analyse :

- **`stops.txt`** : Ce fichier répertorie les arrêts avec leurs coordonnées géographiques et d'autres attributs. Un extrait est présenté dans le tableau 1.1.
- **`routes.txt`** : Il décrit les lignes de transport, avec des informations comme le nom court, le nom long, et le type de transport. Voir le tableau 1.2.
- **`trips.txt`** : Ce fichier associe les trajets aux lignes et aux services. Un exemple est donné dans le tableau 1.3.
- **`stop_times.txt`** : Il contient les horaires d'arrivée et de départ pour chaque arrêt d'un trajet. Voir le tableau 1.4.

TABLEAU 1.1 – Extrait du fichier `stops.txt`

stop_id	stop_name	stop_lat	stop_lon	parent_station
1101064	Malpensa Aeroporto, terminal 1	45.6272	8.7111	
8000339	Weissenhorn Eschach	48.3010	10.1351	
8000709 :0 :2	Neckarsulm Mitte	49.1935	9.2229	
8000778	Asselheim (D)	49.5762	8.1616	
8000781	Grünstadt-Nord	49.5734	8.1708	
8000988	Witzighausen	48.3174	10.0978	
8002015	Nördlingen	48.8508	10.4979	8002015P
8002015 :0 :4	Nördlingen	48.8509	10.4979	8002015P

30. Open Transport Data Swiss. GTFS Cookbook [en ligne]. Disponible sur : <https://opentransportdata.swiss/de/cookbook/timetable-cookbook/gtfs/> (consulté le 2025-01-15).

TABLEAU 1.2 – Extrait du fichier routes.txt

route_id	agency_id	route_short_name	route_desc	route_type
91-10-A-j22-1	37	10	T	900
91-10-B-j22-1	78	S10	S	109
91-10-C-j22-1	11	S10	S	109
91-10-E-j22-1	65	S10	S	109
91-10-F-j22-1	11	RE10	RE	106
91-10-G-j22-1	11	SN10	SN	109
91-10-j22-1	3849	10	T	900
91-10-Y-j22-1	82	IR	IR	103

TABLEAU 1.3 – Extrait du fichier trips.txt

route_id	trip_id	trip_short_name	direction_id
91-8-H-j25-1	994.TA.91-8-H-j25-1.59.R	6278	1
91-8-H-j25-1	995.TA.91-8-H-j25-1.59.R	2978	1
91-8-H-j25-1	996.TA.91-8-H-j25-1.59.R	2787	1
91-8-H-j25-1	997.TA.91-8-H-j25-1.59.R	4879	1
91-8-H-j25-1	998.TA.91-8-H-j25-1.59.R	10407	1
91-8-H-j25-1	999.TA.91-8-H-j25-1.59.R		1

TABLEAU 1.4 – Extrait du fichier stop_times.txt

trip_id	arrival_time	departure_time	stop_id	stop_sequence
1.TA.1-9-j17-1.1.H	05 :25 :00	05 :25 :00	8502034 :0 :2	1
1.TA.1-9-j17-1.1.H	05 :28 :00	05 :29 :00	8502033 :0 :2	2
1.TA.1-9-j17-1.1.H	05 :33 :00	05 :33 :00	8502032 :0 :1	3
1.TA.1-9-j17-1.1.H	05 :36 :00	05 :36 :00	8502031 :0 :1	4
1.TA.1-9-j17-1.1.H	05 :42 :00	05 :42 :00	8502030 :0 :2	5
1.TA.1-9-j17-1.1.H	05 :50 :00	05 :50 :00	8502119 :0 :7	6
2.TA.1-9-j17-1.2.H	05 :53 :00	05 :53 :00	8502034 :0 :1	1
2.TA.1-9-j17-1.2.H	05 :57 :00	05 :58 :00	8502033 :0 :2	2

b. Clés d'identification, normalisation et jointure

Dans le script `get_atlas_data.py`, nous construisons un jeu de données intégré qui associe chaque arrêt aux couples (`route_id`, `direction_id`) desservis et aux noms de lignes, puis relions ces arrêts aux SLOIDs ATLAS. La jointure exploite `stop_times.txt` pour relier les arrêts aux trajets via `trip_id`, puis `trips.txt` et `routes.txt` pour relier ces trajets aux lignes via `route_id`. Nous dédupliquons ensuite les paires route–direction par arrêt et ajoutons le nom de ligne.

Clé de correspondance stop_id GTFS → SLOID ATLAS. Nous associons `stop_id` (GTFS) aux SLOIDs ATLAS via `number` (UIC) et `designation` (référence locale de quai), avec normalisation minimale. Les règles, *telles qu'implémentées dans le code*, sont les suivantes :

- **Structure de stop_id** : `uic_number:0:local_ref`. Exemple : `8516155:0:1`.
- **Strict** : associer si `uic_number = number` et `normalized_local_ref = designation`.
- **Fallback 1** (si non associé strictement) : si le `number` côté ATLAS n'a qu'une seule ligne, utiliser son `sloid`.
- **Fallback 2** (sinon) : si la dernière composante du `sloid` (après les `:`) est égale à `normalized_local_ref`, utiliser ce `sloid`.
- **Normalisation** : les références locales « `10000/10001` » sont ramenées à « `1/2` » lorsqu'elles codent des côtés/plateformes.

La mise en correspondance entre la colonne `stop_id` du fichier `stops.txt` (GTFS) et l'identifiant `sloid` d'ATLAS présente des défis importants. Premièrement, il n'existe pas de lien direct entre ces deux identifiants. Deuxièmement, les coordonnées géographiques des arrêts diffèrent entre les deux ensembles de données.

Exemple 1 : "Lancy-Pont-Rouge" :

Considérons la gare "Lancy-Pont-Rouge", opérée par les CFF. Dans le fichier `stops.txt` de GTFS, les données sont les suivantes :

TABLEAU 1.5 – Extrait de `stops.txt` pour "Lancy-Pont-Rouge"

<code>stop_id</code>	<code>stop_name</code>	<code>stop_lat</code>	<code>stop_lon</code>	<code>parent_station</code>
<code>8516155:0:1</code>	Lancy-Pont-Rouge	46.18596197	6.12483039	Parent8516155
<code>8516155:0:2</code>	Lancy-Pont-Rouge	46.18595575	6.12495615	Parent8516155

Dans le fichier `traffic-points-actual-data`, on trouve :

TABLEAU 1.6 – Extrait de `traffic-points-actual-data` pour "Lancy-Pont-Rouge"

<code>sloid</code>	<code>number</code>	<code>des.</code>	<code>wgs84East</code>	<code>wgs84North</code>	<code>designationOfficial</code>
<code>...:16155:1:1</code>	8516155	1	6.12483137	46.18596333	Lancy-Pont-Rouge
<code>...:16155:1:2</code>	8516155	2	6.12495213	46.18595284	Lancy-Pont-Rouge

Ici, le format de `stop_id` dans GTFS est `uic_number:0:local_ref`, où `uic_number` correspond à la colonne `number` dans ATLAS (8516155), et `local_ref` à `designation` (1 ou 2). Cela permet une correspondance, bien que les coordonnées géographiques divergent légèrement.

Exemple 2 : "Lausanne Bourdonnette" :

Prenons un deuxième exemple avec "Lausanne Bourdonnette". Dans `stops.txt` :

TABLEAU 1.7 – Extrait de stops.txt pour "Lausanne Bourdonnette"

stop_id	stop_name	stop_lat	stop_lon
8501210 :0 :10000	Lausanne, Bourdonnette	46.52342565	6.59074161
8501210 :0 :10001	Lausanne, Bourdonnette	46.52329585	6.58987025
8501210 :0 :A	Lausanne, Bourdonnette	46.52326494	6.58980736
8501210 :0 :B	Lausanne, Bourdonnette	46.52318459	6.58978940
8501210 :0 :C	Lausanne, Bourdonnette	46.52272720	6.58913363
8501210 :0 :D	Lausanne, Bourdonnette	46.52338238	6.59138840

Et dans traffic-points-actual-data :

TABLEAU 1.8 – Extrait de traffic-points-actual-data pour "Lausanne Bourdonnette"

sloid	number	des.	wgs84East	wgs84North	designationOfficial
... :1210 :0 :1600	8501210		6.59074107	46.52342597	Lausanne, Bourdonnette
... :1210 :0 :1610	8501210		6.58986994	46.52329351	Lausanne, Bourdonnette
... :1210 :0 :1616	8501210	B	6.58979344	46.52318499	Lausanne, Bourdonnette
... :1210 :0 :2597	8501210	D	6.59138793	46.52338108	Lausanne, Bourdonnette
... :1210 :0 :2542	8501210	C	6.58913042	46.52272550	Lausanne, Bourdonnette

Dans ce cas, les désignations dans GTFS incluent "A", "B", "C", "D", ainsi que des références numériques comme "10000" et "10001", mais dans ATLAS, "A" n'a pas d'équivalent direct, et les références numériques ne sont pas assignées (lignes avec designation vide). Les coordonnées géographiques diffèrent également.

c. Résultats

Nous obtenons un jeu intégré listant, par sloid, les couples (route_id, direction_id).

Sur notre jeu :

- **SLOIDs couverts par GTFS : 34 415.**
- **Médiane des lignes par SLOID (GTFS) : 2.**

1.3. HRDF

Le format *HAFAS Raw Data Format* (HRDF) structure des jeux de données d'horaires exhaustifs. Deux fichiers sont clés : BHFART (SLOIDs gare/quai) et GLEISE_WGS/LV95 (infrastructure et coordonnées de quai).

Référence. Une description synthétique et à jour du format HRDF est disponible dans le Cook-book de la plateforme Open Transport Data Switzerland³².

32. Open Transport Data Swiss. HRDF Cookbook [en ligne]. Disponible sur : <https://opentransportdata.swiss/en/cookbook/timetable-cookbook/hafas-rohdaten-format-hrdf/> (consulté le 2025-08-17).

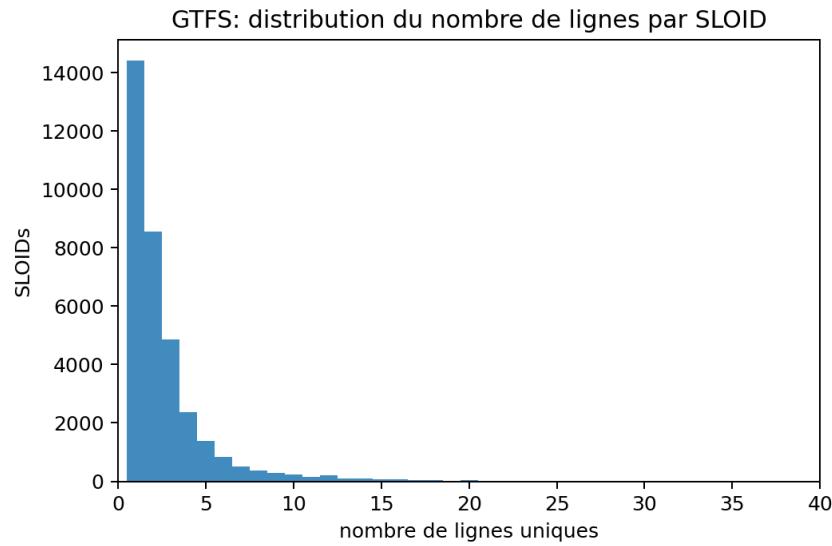


ILLUSTRATION 1.3 – GTFS : distribution du nombre de lignes par SLOID.

a. Comment nous le lisons

L'extraction des informations de direction à partir des données HRDF est un processus en plusieurs étapes, orchestré par le script `get_atlas_data.py`. Notre approche est conçue pour être efficace, en ne traitant que les données pertinentes pour les arrêts ATLAS (SLOIDs) que nous cherchons à enrichir.

1. **Identification des voyages par quai (GLEISE_LV95).** La première étape consiste à identifier les voyages (trajets) qui desservent chaque quai d'intérêt. Pour ce faire, nous parcourons le fichier `GLEISE_LV95` en deux passes :
 - D'abord, nous établissons une correspondance entre chaque `sloid` de quai et son couple identifiant (`UIC de gare, référence de quai`). La référence de quai est un identifiant local, souvent préfixé par un #.
 - Ensuite, nous relisons le même fichier pour trouver toutes les occurrences de ces paires (`UIC, #ref`), ce qui nous permet de collecter l'ensemble des voyages (identifiés par un numéro de voyage et un numéro d'opérateur) qui s'arrêtent à ces quais. Cette approche ciblée évite de charger en mémoire la totalité des voyages, qui est extrêmement volumineuse.
2. **Extraction des directions de voyage (FPLAN).** Une fois que nous avons la liste des voyages pour chaque quai, nous analysons le fichier `FPLAN`. Ce fichier décrit la séquence des arrêts pour chaque voyage. Pour chaque voyage d'intérêt, nous extrayons :
 - Le nom de la ligne (information préfixée par *L).
 - Le premier et le dernier arrêt du trajet (codes UIC).
 La séquence premier/dernier arrêt nous donne la direction du voyage.

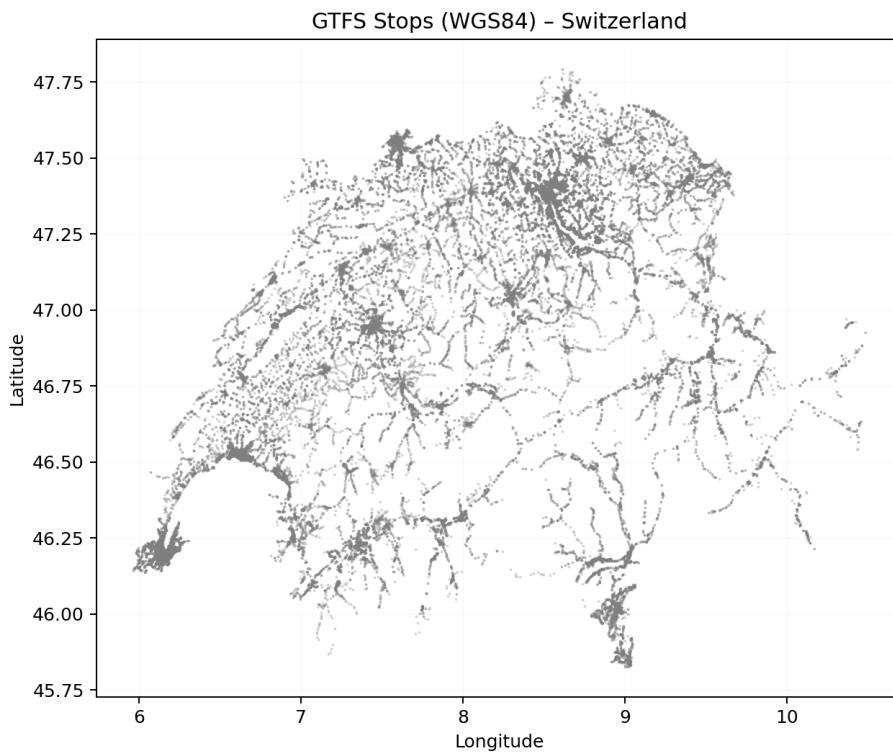


ILLUSTRATION 1.4 – Arrêts GTFS (Suisse, à l'intérieur de la frontière) : 47 567 arrêts détectés.

3. **Association des noms de gares (BAHNHOF).** Les codes UIC extraits de FPLAN sont des identifiants numériques. Pour les rendre lisibles, nous utilisons le fichier BAHNHOF qui sert de dictionnaire pour faire correspondre chaque code UIC à un nom de gare en toutes lettres.
4. **Construction des chaînes de direction.** Enfin, nous assemblons toutes ces informations. Pour chaque `sloid`, nous créons des chaînes de direction uniques en combinant les noms de ligne et les directions. Par exemple, un voyage peut être résumé par une direction textuelle comme "Genève → Lausanne" et une direction UIC comme "8501008 → 8501120". Ces informations enrichissent nos données d'arrêts ATLAS avec des contextes d'itinéraire précieux.

b. Informations exploitées

- **SLOID de quai** et position (WGS84);
- **Chaînes directionnelles** *nom* (*Genève → Lausanne*) et *UIC* (8501008 → 8501120).

c. Statistiques clés

Compte les SLOIDs de quai uniques référencés dans GLEISE_LV95.

Commande

```
1 $ grep -o "g A ch:1:sloid:[^[:space:]]\\\" data/raw/GLEISE_LV95 | \\
2   sed 's/^g A //\' | sort -u | wc -l
3 30935
```

Compte les SLOIDs de quai uniques dans BHFART (entrées « G a »).

Commande

```
1 $ grep -o "G a ch:1:sloid:[^[:space:]]\\\" data/raw/BHFART | sed 's/^G a //\' | sort
- u | wc -l
2 30935
```

Compte les SLOIDs de gare uniques dans BHFART (entrées « G A »).

Commande

```
1 $ grep -o "G A ch:1:sloid:[^[:space:]]\\\" data/raw/BHFART | sed 's/^G A //\' | sort
- u | wc -l
2 31913
```

Après extraction des informations de direction : 28723 SLOIDs couverts ; médiane des directions (noms) par SLOID : **4**. Ces chaînes « nom » et « UIC » enrichissent les correspondances lorsqu'un identifiant de direction explicite n'est pas disponible côté OSM.

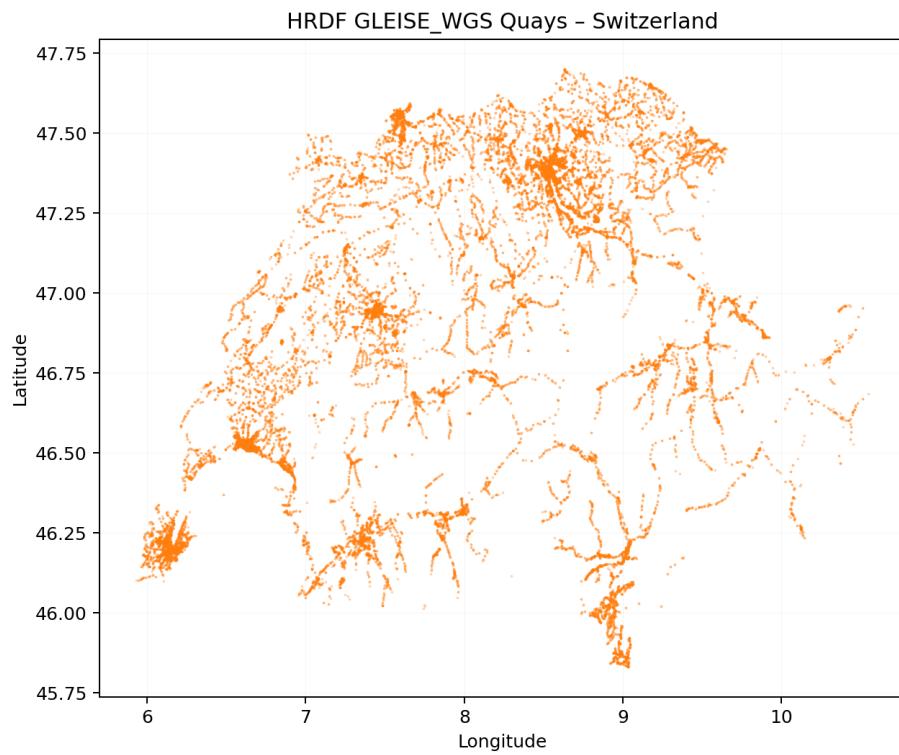


ILLUSTRATION 1.5 – Quais HRDF (GLEISE_WGS) – Suisse.

1.4. COMPARAISON GTFS ET HRDF (COUVERTURE SLOID)

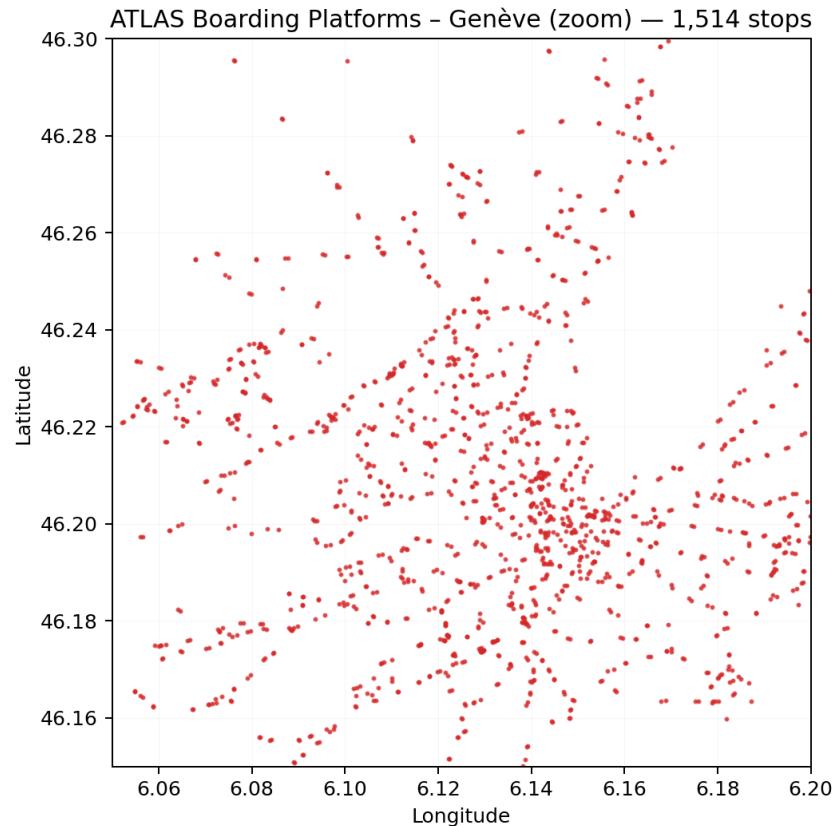


ILLUSTRATION 1.6 – ATLAS – plateformes d'embarquement, zoom Genève.

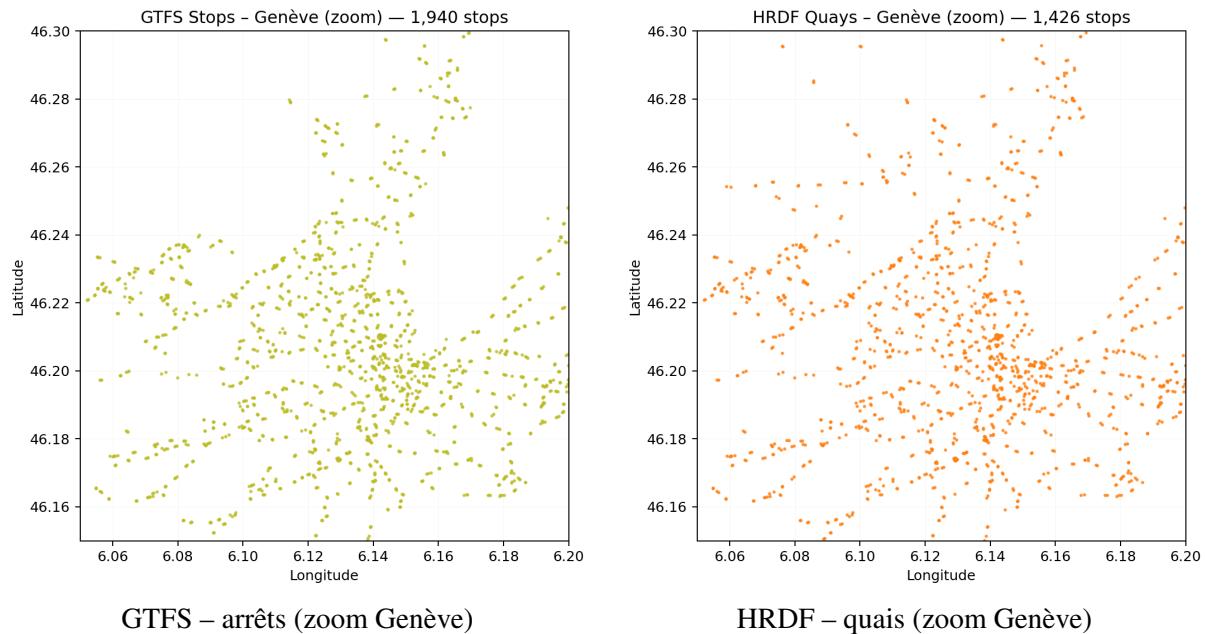


ILLUSTRATION 1.7 – Genève : comparaison des points bruts GTFS (gauche) et HRDF (droite).

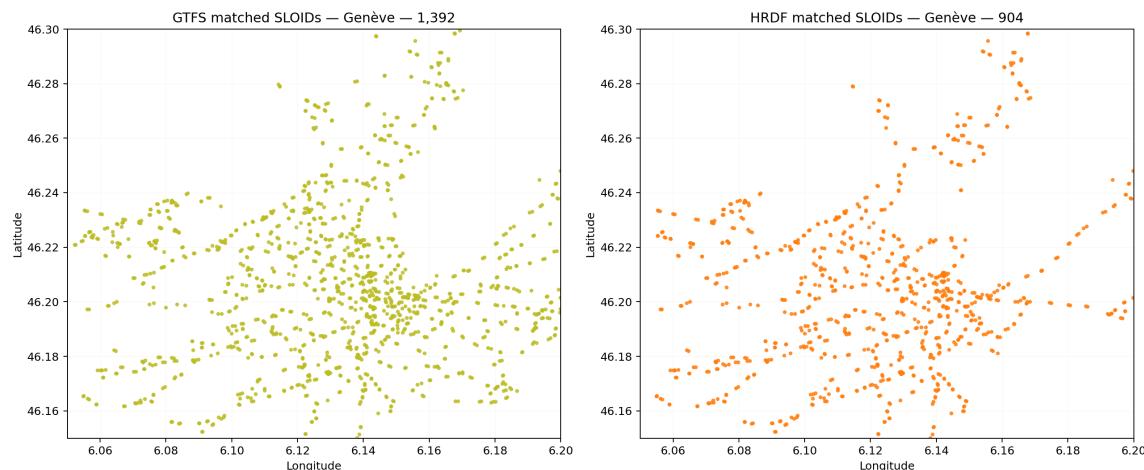


ILLUSTRATION 1.8 – Genève : SLOIDs ATLAS couverts par les jeux intégrés **GTFS** (gauche) et **HRDF** (droite).

Concernant la couverture globale des SLOIDs ATLAS par les deux jeux de données intégrés, nous observons les statistiques suivantes :

- GTFS : 34415 SLOIDs ; HRDF : 28723 SLOIDs.
- Intersection : 14224 ; GTFS seulement : 20191 ; HRDF seulement : 14499.

CHAPITRE 2 : TRANSPORT PUBLIC DANS OSM

La cartographie du transport public dans OpenStreetMap (OSM) a évolué à travers plusieurs schémas. Cette évolution a conduit à la coexistence de diverses combinaisons de balises pour les arrêts de bus, gares ferroviaires, stations de tramway et autres nœuds de transport. De plus, comme OSM est un projet maintenu par une communauté de volontaires, certaines entrées peuvent ne correspondre à aucun schéma précis.

Dans cette section, nous analysons les différents schémas existants, nous présentons la requête utilisée (c'est-à-dire quelles données seront extraites) et, une fois les données obtenues, nous proposons une vue d'ensemble de l'usage des balises pour les nœuds d'arrêts de transport public dans OSM en Suisse.

Reproductibilité. Les figures et statistiques de ce chapitre sont produites par des scripts sous `memoire/scripts_used/chap2/`. Le script principal est `osm_plots.py` et lit `data/-raw/osm_data.xml` ainsi que les fichiers traités générés par `get_osm_data.py`.

2.1. SCHÉMAS DE CARTOGRAPHIE DU TRANSPORT PUBLIC DANS OSM

- **Schéma d'origine (PTv1)** : La méthode la plus ancienne et encore très répandue, qui attribue à chaque arrêt des balises spécifiques au mode concerné. Par exemple, un arrêt de bus est simplement `highway=bus_stop`⁵, une gare ferroviaire est `railway=station` (ou `railway=halt` pour des arrêts plus petits), et un arrêt de tramway est `railway=tram_stop`. Ces balises figurent souvent sur un seul nœud représentant l'emplacement où les passagers attendent. PTv1 est largement utilisé encore aujourd'hui⁶. Il est important de noter qu'aucune de ces anciennes balises n'a été formellement dépréciée par les propositions plus récentes, ce qui explique qu'elles restent toujours en usage actif.
- **Schéma Oxomoa (années 2010)** : Schéma intermédiaire développé vers 2010 (par l'utilisateur Oxomoa), il introduisait une structure plus aboutie, ressemblant à ce que PTv2 allait proposer plus tard. Ce schéma utilisait des relations de type "route" et des relations de type "stop area" pour regrouper les éléments d'arrêt⁶. Bien qu'il ait influencé la version suivante, ce schéma est désormais historique, même si certains itinéraires plus anciens (~2010) le suivent encore.
- **Nouveau schéma de transport public (PTv2)** : Approuvé en 2011, PTv2 a introduit un système de balisage plus puissant mais plus complexe⁸. L'idée est de séparer la

5. OpenStreetMap Wiki. DE :Tag :highway=bus_stop [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/DE:Tag:highway=bus_stop (consulté le 2025-07-14).

6. OpenStreetMap Wiki. Public transport [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Public_transport (consulté le 2025-07-15).

8. OpenStreetMap Wiki. Proposal :Public transport schema [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Proposal:Public_transport_schema.

notion d'arrêt en stop positions (là où le véhicule s'arrête sur la chaussée ou la voie) et platforms (où les passagers attendent). Dans ce schéma, un arrêt de bus est généralement représenté par *deux* objets reliés :

- un nœud sur la chaussée avec `public_transport=stop_position` (souvent accompagné de `bus=yes` ou `tram=yes`, etc., pour préciser le mode)⁷,
- et un nœud (ou une zone) en bord de route portant la balise `public_transport=platform` (en plus d'une balise pour le mode ou d'une balise héritée).

Par exemple, un nœud de plate-forme de bus peut porter `public_transport=platform + bus=yes`, tandis que le nœud correspondant sur la chaussée sera `public_transport=stop_position + bus=yes`⁷. En pratique, les cartographes incluent souvent l'ancienne balise sur l'un de ces objets pour assurer la compatibilité – par exemple, on retrouvera `highway=bus_stop` sur le nœud de la plate-forme, afin qu'il soit reconnu par les outils traditionnels⁵.

PTv2 introduit également la notion de relation `stop_area` (`type=public_transport + public_transport=stop_area`) pour regrouper tous les éléments d'une même station ou d'un même arrêt, et une relation `route_master` pour regrouper les itinéraires dans les deux sens⁸. Fait notable, la proposition PTv2 n'a pas invalidé ni remplacé les balises existantes, ce qui signifie que les balises PTv1 (telles que `highway=bus_stop`, `railway=station`) coexistent souvent avec les balises PTv2 pour un même arrêt⁶. De nombreuses communautés encouragent à ajouter les balises PTv2 tout en conservant les anciennes pour plus de complétude.

Remarque (zones). Pour ce projet, nous ne considérons que les **nœuds**. Par simplification, les *plateformes* ou *stop_positions* modélisées comme des **zones** (ways/relations) ne sont pas intégrées au comptage. Il peut exister des `public_transport=platform` ou `public_transport=stop_position` cartographiés en zones.

2.2. DIFFÉRENCES DANS L'USAGE DE CLÉS SPÉCIFIQUES

Certains choix de clés varient parmi les cartographes, ce qui peut engendrer des divergences dans la manière de consigner l'information :

- `ref` vs `local_ref` (codes d'arrêt) : De nombreux arrêts de transport public possèdent

openstreetmap.org/wiki/Proposal:Public_transport_schema (consulté le 2025-07-17).

7. OpenStreetMap Wiki. FR :Key :public_transport [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/FR:Key:public_transport (consulté le 2025-07-16).

5. OpenStreetMap Wiki. DE :Tag :highway=bus_stop [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/DE:Tag:highway=bus_stop (consulté le 2025-07-14).

8. OpenStreetMap Wiki. Proposal :Public transport schema [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Proposal:Public_transport_schema (consulté le 2025-07-17).

6. OpenStreetMap Wiki. Public transport [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Public_transport (consulté le 2025-07-15).

un code ou un identifiant officiel (numéro ou lettre fourni par l'autorité de transport). Les cartographes utilisent tantôt la balise générique `ref=`, tantôt `local_ref=`. La recommandation d'OSM est d'utiliser `ref=` pour le code d'arrêt à l'échelle du réseau (un ID unique dans le système de transport) et `local_ref` si c'est un code ou une lettre propre à un contexte plus restreint¹⁰.

Par exemple, un arrêt de bus qui a l'ID « 3154 » dans la base de données de la ville se balisera `ref=3154`. Si cet arrêt comporte plusieurs quais, nommés « Bay C » par exemple, on peut utiliser `local_ref=C` sur le quai concerné. En pratique, la distinction n'est pas toujours respectée : certains mettent tous les codes dans `ref`, d'autres utilisent `local_ref` pour les numéros de quai ou les lettres d'arrêt.

2.3. REQUÊTE OVERPASS TRANSPORT PUBLIC — SUISSE

Overpass est un système de requêtage permettant d'extraire des données de la base de données OpenStreetMap¹¹. Il utilise un langage de requête appelé Overpass Query Language, qui permet de rechercher et de filtrer des objets OSM (nœuds, chemins, relations) en fonction de critères spécifiques (tags, zones géographiques, types d'objets, etc.). Pour obtenir les arrêts de transport public en Suisse sur OpenStreetMap, nous utilisons la requête Overpass suivante (simplifiée et *dédoublonnée*) :

Requête Overpass

```
[out:xml][timeout:180];
area["ISO3166-1"="CH"]->.searchArea;
(
  node(area.searchArea)["public_transport"~"platform|stop_position"];
  node(area.searchArea)["highway"="bus_stop"];
  node(area.searchArea)["railway"~"station|halt|tram_stop"];
  node(area.searchArea)["amenity"~"bus_station|ferry_terminal"];
  node(area.searchArea)["aerialway"="station"];
);
out;
relation(bn)[type=route];
out meta;
```

Cette requête commence par définir la zone d'intérêt, qui correspond à la Suisse, identifiée par son code ISO3166-1 CH. Ensuite, elle sélectionne différents types de nœuds correspondant aux infrastructures de transport public. Cette requête inclut des arrêts de bus et de tram, des

10. OpenStreetMap Wiki. Key :local_ref [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Key:local_ref (consulté le 2025-07-19).

11. Overpass Turbo [en ligne]. Disponible sur : <https://overpass-turbo.eu/> (consulté le 2025-07-20).

terminaux de ferries, des stations de remontées mécaniques, etc.

Enfin, la requête extrait également les relations de type `route` associées aux nœuds obtenus. Cette information est pertinente, car elle permet de lier les arrêts à leurs itinéraires respectifs, ce qui facilitera les correspondances avec d'autres sources de données, comme les données ATLAS.

2.4. APERÇU DES BALISES (*tags*) DES NŒUDS OSM EN SUISSE

Une fois les nœuds obtenus par la requête ci-dessus, nous analysons les balises qu'ils contiennent. Nous montrons d'abord quelques exemples :

OSM Nœud : Grand-Mont

```
<node id="2368323780" lat="46.5627599" lon="6.6343369">
  <tag k="bus" v="yes"/>
  <tag k="highway" v="bus_stop"/>
  <tag k="local_ref" v="D"/>
  <tag k="name" v="Grand-Mont"/>
  <tag k="network" v="Mobilis"/>
  <tag k="operator" v="TL"/>
  <tag k="public_transport" v="stop_position"/>
  <tag k="tactile_paving" v="no"/>
  <tag k="trolleybus" v="yes"/>
  <tag k="uic_name" v="Le Mont-sur-L., Grand-Mont"/>
  <tag k="uic_ref" v="8504177"/>
</node>
```

OSM Nœud sans nom

```
<node id="2368860496" lat="46.4418646" lon="6.9764107">
  <tag k="aerialway" v="station"/>
</node>
```

OSM Nœud : Interlaken Ost

```
</node>
<node id="2388274179" lat="46.6910098" lon="7.8697428">
  <tag k="name" v="Interlaken Ost"/>
  <tag k="public_transport" v="stop_position"/>
  <tag k="railway" v="stop"/>
  <tag k="ref" v="7"/>
  <tag k="train" v="yes"/>
</node>
```

Comme on peut le constater, chaque nœud contient des balises différentes. Voici quelques statistiques pour une vision générale (sur notre extraction) :

- Nombre total de nœuds : 60 635
- Nombre total de nœuds avec `public_transport == platform` : 24 548
 - Parmi ceux-ci avec `uic_ref` : 22 986
 - Nœuds de plateforme avec une position d'arrêt correspondante (même `uic_ref`) : 13 571
 - Nœuds de plateforme avec `uic_ref` mais sans position d'arrêt correspondante : 9 415
- Nombre total de nœuds avec `public_transport == stop_position` : 30 018
 - Parmi ceux-ci avec `uic_ref` : 28 199
- Nœuds avec toutes les balises (`uic_ref`, `local_ref`, `name`, `network`, `operator`, `uic_name`) : 3 875
- Nœuds avec `uic_ref` : 55 166
 - Parmi ceux-ci, avec `ref` : 2 796
 - Parmi ceux-ci, avec `local_ref` : 4 314
 - Parmi ceux-ci, avec `ref` et `local_ref` : 307
 - Parmi ceux-ci avec `name` : 55 147
 - Parmi ceux-ci avec `network` : 40 576
 - Parmi ceux-ci avec `operator` : 53 322
 - Parmi ceux-ci avec `uic_name` : 55 042
- Nœuds sans `uic_ref` : 5 469
 - Parmi ceux-ci, avec `ref` : 200
 - Parmi ceux-ci, avec `local_ref` : 288
 - Parmi ceux-ci, avec `ref` et `local_ref` : 13
 - Parmi ceux-ci avec `name` : 4 339
 - Parmi ceux-ci avec `network` : 758
 - Parmi ceux-ci avec `operator` : 1 542
 - Parmi ceux-ci avec `uic_name` : 86
- Nombre total de nœuds sans aucune des balises `uic_ref`, `ref`, `local_ref`, `network`, `operator`, `uic_name` : 1 084
- Nœuds non assignés avec `aerialway=station` : 817

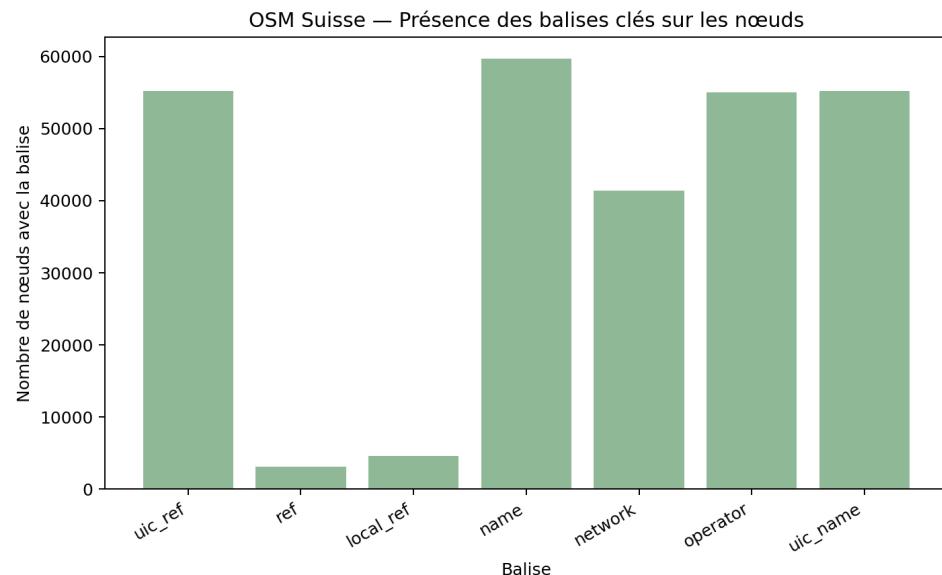


ILLUSTRATION 2.1 – Présence des balises clés par nœud (uic_ref, ref, local_ref, name, network, operator, uic_name).

2.5. APERÇU DES ITINÉRAIRES DE TRANSPORT PUBLIC DANS OSM EN SUISSE

Comme mentionné dans le chapitre 1, nous nous intéressons également aux itinéraires, car ils peuvent nous aider à identifier des correspondances. Cela est particulièrement utile lorsqu'il existe deux arrêts pour une même station, mais pour des itinéraires empruntant des directions opposées, ou lorsque des arrêts de bus et de tram sont situés à proximité.

Voici quelques statistiques essentielles :

- Total d'itinéraires uniques : 1 904
- Total de connexions entre nœuds et itinéraires : 138 761
- Nombre de nœuds desservant au moins un itinéraire : 51 286
- Nombre moyen d'itinéraires par nœud : 2,71

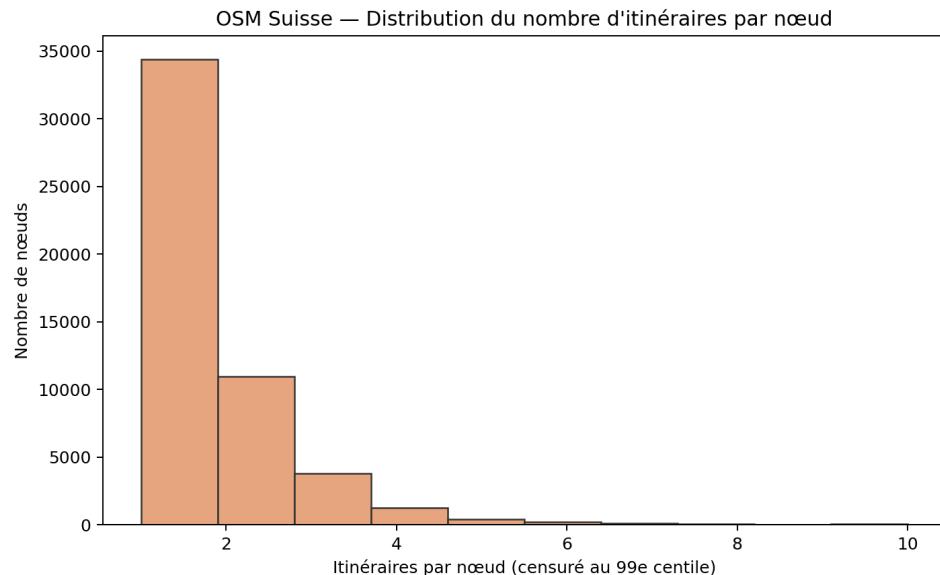


ILLUSTRATION 2.2 – OSM (Suisse) — distribution du nombre d'itinéraires distincts par nœud (censurée au 99e centile).

a. Les 5 nœuds de transport public les plus « connectés »

Note méthodologique. Cette liste est calculée en comptant le nombre de **connexions nœud-itinéraire** présentes dans le fichier `data/processed/osm_nodes_with_routes.csv` (une ligne par appartenance d'un nœud à une relation OSM de type `route`). Il s'agit donc d'un **compte brut des relations** (directions et variantes incluses), et non d'un décompte de lignes distinctes après déduplication par `gtfs:route_id`. Le script minimal reproduisant ce calcul est fourni dans `memoire/scripts_used/chap2/compute_busiest_nodes.py`.

1er — Zürich Bus Station

Itinéraires desservis : 65

Type de nœud : stop_position

Node ID : 5962551000

2e — Stein

Itinéraires desservis : 40

Type de nœud : stop_position

Référence UIC : 8580638

Node ID : 984028248

3e — Genève - Gare Routière

Itinéraires desservis : 37

Type de nœud : stop_position

Node ID : 960890428

4e — Lugano Centrale

Itinéraires desservis : 37

Type de nœud : stop_position

Référence UIC : 8505550

Node ID : 984002736

5e — Paradiso

Itinéraires desservis : 34

Type de nœud : stop_position

Référence UIC : 8505553

Node ID : 1266983076

b. Analyse des directions des itinéraires

- Direction 0 (généralement sortante) : 60 319 connexions
- Direction 1 (généralement entrante) : 57 057 connexions
- Direction inconnue : 21 385 connexions

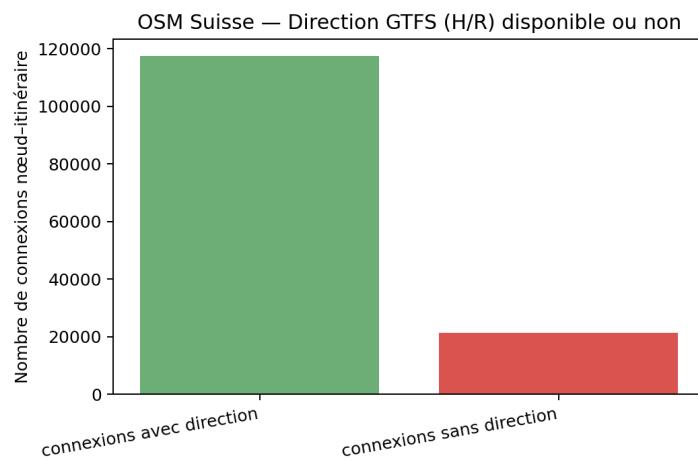


ILLUSTRATION 2.3 – Connexions noeud–itinéraire pour lesquelles une direction (H/R) est déduite à partir de ref_trips.

c. Top 5 des itinéraires avec le plus d’arrêts

- Bus 120 : Engelburg → St. Gallen → Eggersriet → Heiden : 174 arrêts

- Bus 120 : Heiden → Eggersriet → St. Gallen → Engelburg : 174 arrêts
- Bus 722 : Weinfelden → Hosenruck → Wil SG : 150 arrêts
- Bus 722 : Wil SG → Hosenruck → Weinfelden : 150 arrêts
- Bus 507 : Lostorf → Olten → Egerkingen : 138 arrêts

CHAPITRE 3 : CORRESPONDANCE AVEC LES DONNÉES ATLAS-OSM

Le processus de correspondance entre les données ATLAS et OSM a été conçu pour identifier de manière précise et systématique les arrêts correspondants dans ces deux ensembles de données. Cette approche méthodologique repose sur un principe d'appariement séquentiel par ordre de fiabilité.

3.1. APPROCHE MÉTHODOLOGIQUE GÉNÉRALE

Le processus de correspondance adopte une stratégie « hit-first » (première correspondance trouvée), où chaque entrée ATLAS est appariée selon la première méthode qui réussit, en suivant un ordre décroissant de fiabilité. Cette approche séquentielle garantit que les correspondances les plus fiables (exactes) sont privilégiées par rapport aux correspondances moins certaines (par distance).

Nous avons envisagé une approche alternative consistant à exécuter toutes les méthodes sur toutes les entrées, afin d'analyser pour chaque correspondance le nombre de méthodes qui fonctionnent et d'attribuer une probabilité de correspondance. Cependant, après réflexion approfondie, nous avons conclu que cette approche n'apporterait pas de valeur ajoutée significative. En effet, l'ordre séquentiel reflète déjà la hiérarchie de fiabilité : si une correspondance exacte est trouvée, il est inutile de vérifier si d'autres méthodes moins fiables fonctionnent également.

Le processus complet comprend les étapes suivantes :

1. **Appariement exact** : Appariement basé sur les identifiants UIC et la référence locale.
2. **Correspondance par nom** : Utilisation des noms officiels des arrêts.
3. **Correspondance par distance** : Analyse géographique avec critères de proximité.
4. **Correspondance par routes** : Méthode complexe basée sur l'analyse des itinéraires (détalée au chapitre 4).
5. **Consolidation post-traitement** : Deuxième passage basé sur les identifiants UIC et les références locales avec les entrées restantes.
6. **Propagation des duplicitas** : Si une entrée ATLAS est dupliquée, la correspondance est propagée à toutes les entrées dupliquées.
7. **Correspondance manuelle** : Application des correspondances définies manuellement et stockées de manière persistante.

Ce chapitre détaille les méthodes de correspondance exacte, par nom et par distance. La correspondance par routes, en raison de sa complexité particulière, sera traitée séparément au chapitre 4.

3.2. CORRESPONDANCE EXACTE

La première étape, appelée correspondance exacte, utilise l'identifiant UIC. Dans les données ATLAS, cet identifiant est représenté par la colonne 'number', tandis que dans OSM, il correspond à la balise 'uic_ref'. Une entrée ATLAS est appariée à un nœud OSM si son 'number' est identique au 'uic_ref' du nœud OSM.

Des situations complexes peuvent survenir lorsque plusieurs entrées ATLAS partagent le même 'number' (par exemple, plusieurs quais d'une même gare) ou lorsque plusieurs nœuds OSM possèdent le même 'uic_ref'. Pour résoudre ces cas, les règles suivantes sont appliquées :

1. **Cas 1 : Plusieurs entrées ATLAS, un seul nœud OSM** Si plusieurs entrées ATLAS partagent le même 'number' et qu'un seul nœud OSM possède ce 'uic_ref', toutes ces entrées ATLAS sont appariées à ce nœud OSM unique.
2. **Cas 2 : Une entrée ATLAS, plusieurs nœuds OSM** Si une seule entrée ATLAS a un 'number' donné et que plusieurs nœuds OSM partagent ce 'uic_ref', tous ces nœuds OSM sont appariées à cette entrée ATLAS unique.
3. **Cas 3 : Plusieurs entrées ATLAS et plusieurs nœuds OSM** Lorsque plusieurs entrées ATLAS et nœuds OSM partagent le même 'number'/'uic_ref', une correspondance plus fine est réalisée en comparant la 'designation' de l'entrée ATLAS (par exemple, le code du quai) avec la balise 'local_ref' du nœud OSM. Une correspondance est établie si ces valeurs sont identiques.

Cette méthode a permis d'identifier 21237 correspondances exactes.

3.3. CORRESPONDANCE PAR NOM

Pour les entrées ATLAS non appariées lors de l'étape précédente, une correspondance basée sur le nom est appliquée. Cette étape compare le nom officiel des arrêts, indiqué dans la colonne 'designationOfficial' des données ATLAS, avec plusieurs balises de nom dans OSM : 'name', 'uic_name' et 'gtfs:name'.

La règle principale établit une correspondance si le 'designationOfficial' correspond exactement à l'une de ces balises OSM. Cependant, si plusieurs nœuds OSM présentent le même nom, un critère supplémentaire est utilisé : la balise 'local_ref' du nœud OSM est comparée à la 'designation' de l'entrée ATLAS. Une correspondance est confirmée si ces valeurs sont identiques (en ignorant la casse).

Cette approche a permis d'ajouter 537 correspondances supplémentaires.

3.4. CORRESPONDANCE PAR DISTANCE

Pour les entrées ATLAS restantes, une correspondance basée sur la proximité géographique est mise en œuvre. Cette étape se divise en trois sous-étapes distinctes, chacune avec des critères

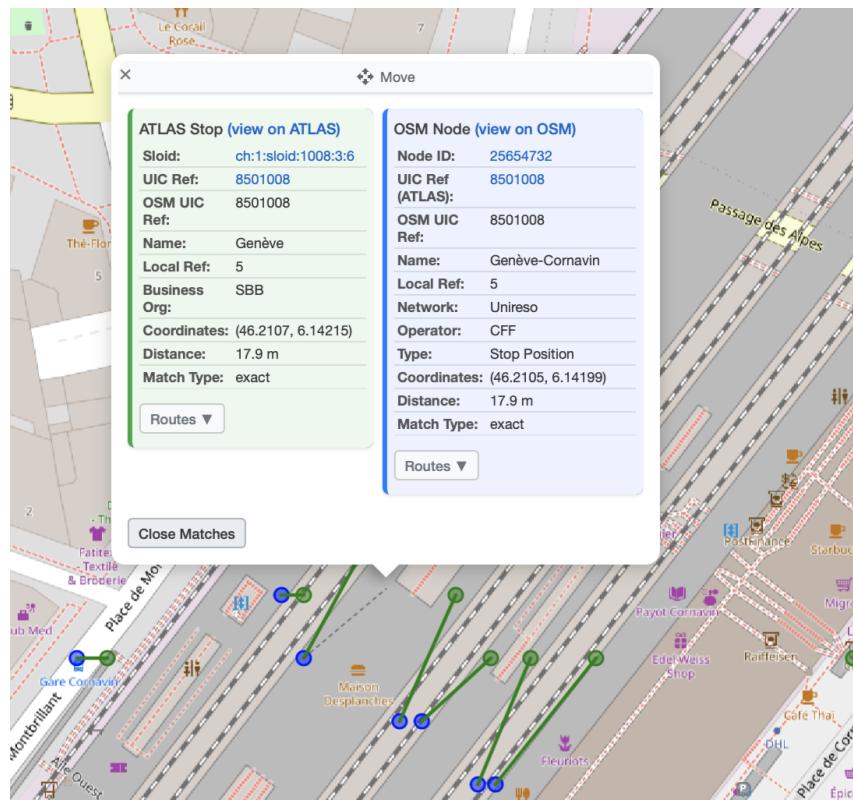


ILLUSTRATION 3.1 – Correspondances exactes à la gare de Genève-Cornavin. Les détails de l’arrêt de la voie 5 sont visibles sur l’image.

spécifiques pour garantir des appariements fiables.

a. Étape 1 : Correspondance de groupe basée sur la proximité

Les entrées ATLAS et OSM sont regroupés selon les paires d’identifiants suivantes :

1. 'number' (ATLAS) et 'uic_ref' (OSM).
2. 'designationOfficial' (ATLAS) et 'uic_name' (OSM).
3. 'designationOfficial' (ATLAS) et 'name' (OSM).

Dans chaque groupe où le nombre d’entrées ATLAS est égal au nombre de nœuds OSM, une correspondance est tentée en associant chaque entrée ATLAS au nœud OSM le plus proche, à condition que cette association soit cohérente (c'est-à-dire que chaque nœud OSM soit également le plus proche de l’entrée ATLAS qui lui est attribuée). Cette méthode nous a permis de réaliser 15 174 correspondances supplémentaires.

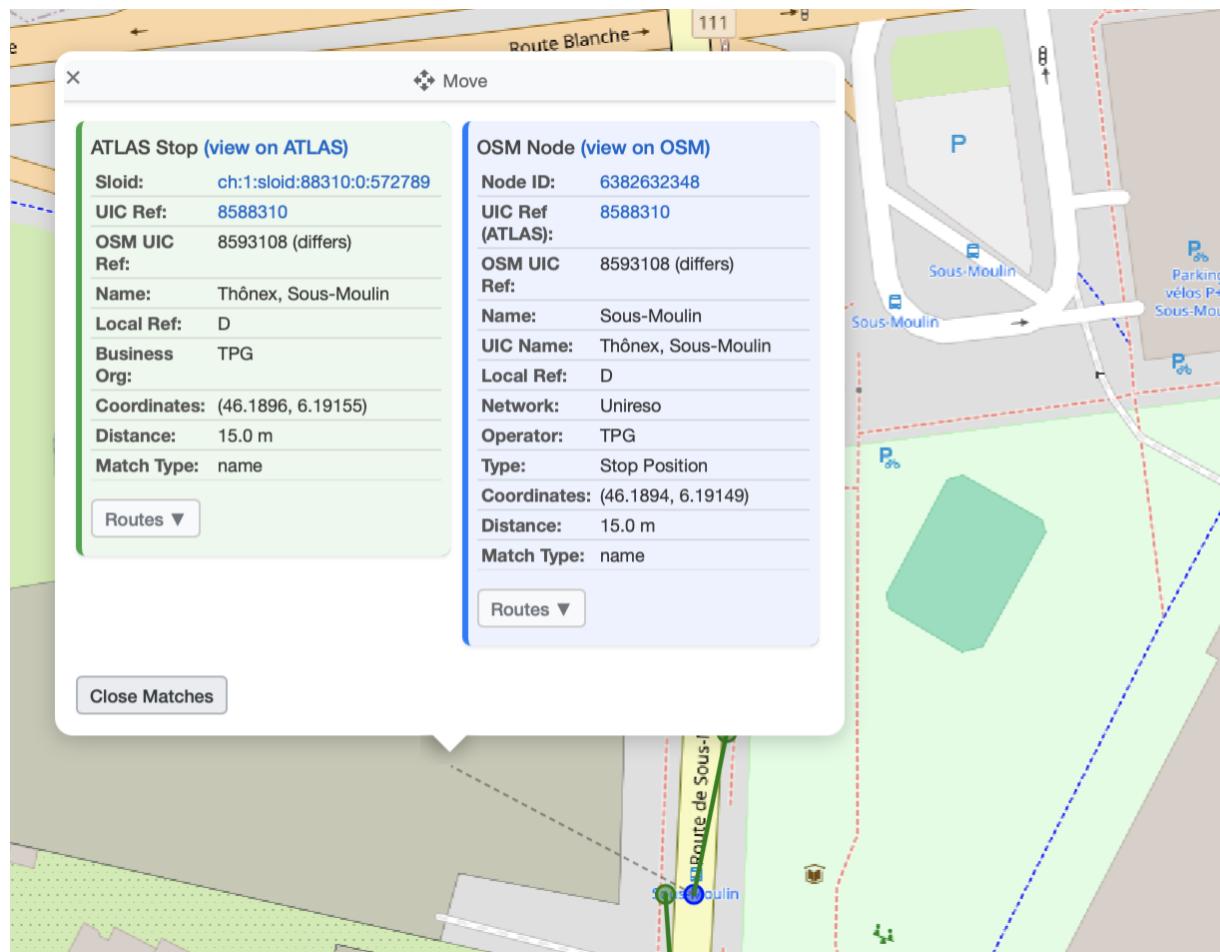


ILLUSTRATION 3.2 – Pour l'arrêt "Thônex, Sous-Moulin, D", on peut voir que, malgré une référence UIC différente, il est possible d'établir des correspondances grâce au nom.

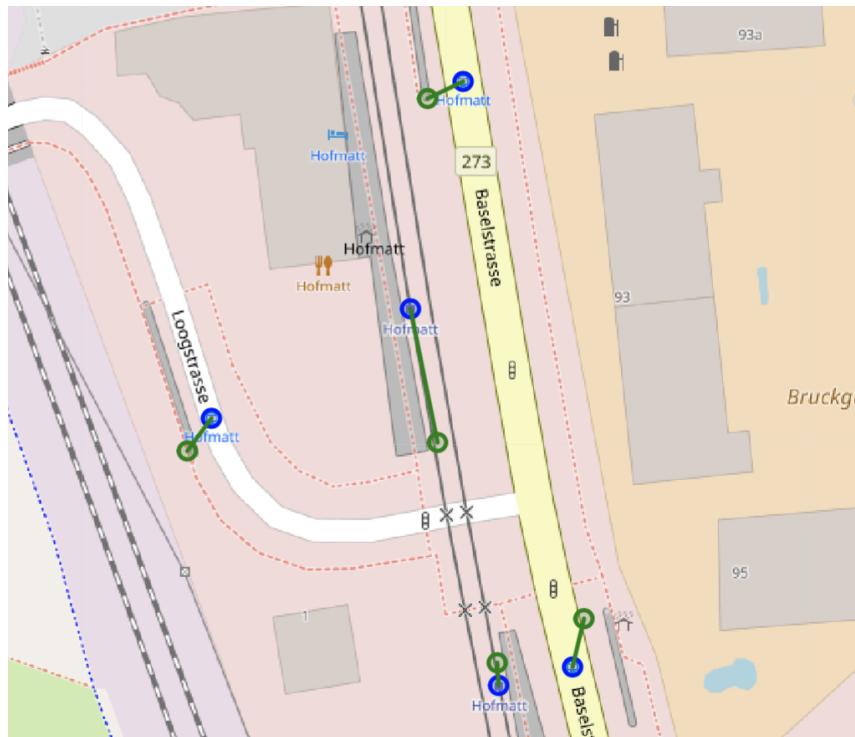


ILLUSTRATION 3.3 – Correspondances pour les arrêts de Münchenstein, Hofmatt. Malgré les divergences de uic_ref et le manque de références locales, nous avons réussi à établir des correspondances grâce à la correspondance de groupe basée sur les distances.

TABLEAU 3.1 – Données ATLAS pour les arrêts de Münchenstein, Hofmatt

sloid	number	designation	designationOfficial
ch :1 :sloid :95 :1 :6	8500095		Münchenstein, Hofmatt
ch :1 :sloid :95 :1 :5	8500095		Münchenstein, Hofmatt
ch :1 :sloid :95 :1 :3	8500095		Münchenstein, Hofmatt
ch :1 :sloid :95 :1 :2	8500095		Münchenstein, Hofmatt
ch :1 :sloid :95 :1 :1	8500095		Münchenstein, Hofmatt

TABLEAU 3.2 – Données OSM pour les arrêts de Münchenstein, Hofmatt

node_id	uic_ref	uic_name	transport_type
6457499611	8578185	Münchenstein, Hofmatt	bus
299126238	8500095	Münchenstein, Hofmatt	tram
983964446	8578185	Münchenstein, Hofmatt	bus
1435404358	8500095	Münchenstein, Hofmatt	tram
3858822225	8578185	Münchenstein, Hofmatt	bus

b. Étape 2 : Correspondance par référence locale dans un rayon de 50 mètres

Cette sous-étape recherche, pour chaque entrée ATLAS non appariée, un nœud OSM situé à moins de 50 mètres dont la balise `local_ref` correspond exactement à la designation de l'entrée ATLAS (en ignorant la casse).

À Zürich HB, dans ATLAS, la `UIC_ref` est égale à 8503000 pour tous les arrêts, tandis que dans OSM, certains arrêts ont une `UIC_ref` de 8516144.

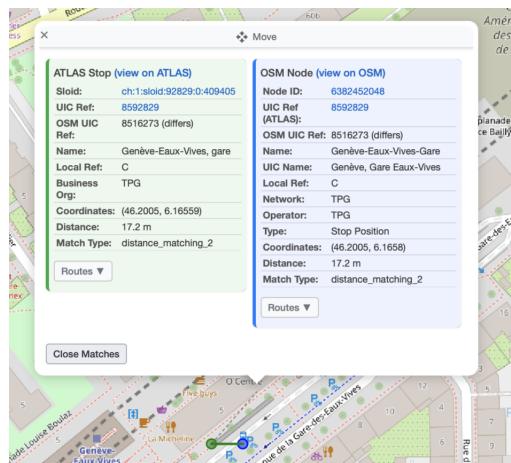


ILLUSTRATION 3.4 – Correspondances à Genève-Eaux-Vives-Gare grâce à l'étape 2.

Cette méthode nous a permis de réaliser 129 correspondances supplémentaires.

c. Étape 3 : Correspondance basée sur la proximité avec critères relatifs

Pour les entrées toujours non appariées, tous les nœuds OSM situés à moins de 50 mètres sont examinés :

- a) Si un seul nœud OSM se trouve dans ce rayon, il est apparié à l'entrée ATLAS.
- Si plusieurs nœuds OSM sont présents, l'appariement est effectué avec le nœud le plus proche uniquement si :
 1. b) Le deuxième nœud le plus proche est à au moins 10 mètres.
 2. La distance au deuxième nœud le plus proche est au moins 4 fois supérieure à celle du nœud le plus proche.

Nous avons réussi à établir 2 123 correspondances avec l'option a) et 1 192 correspondances avec l'option b). Cette méthode est utile pour les cas où il y a des nœuds isolés, comme des télésièges.

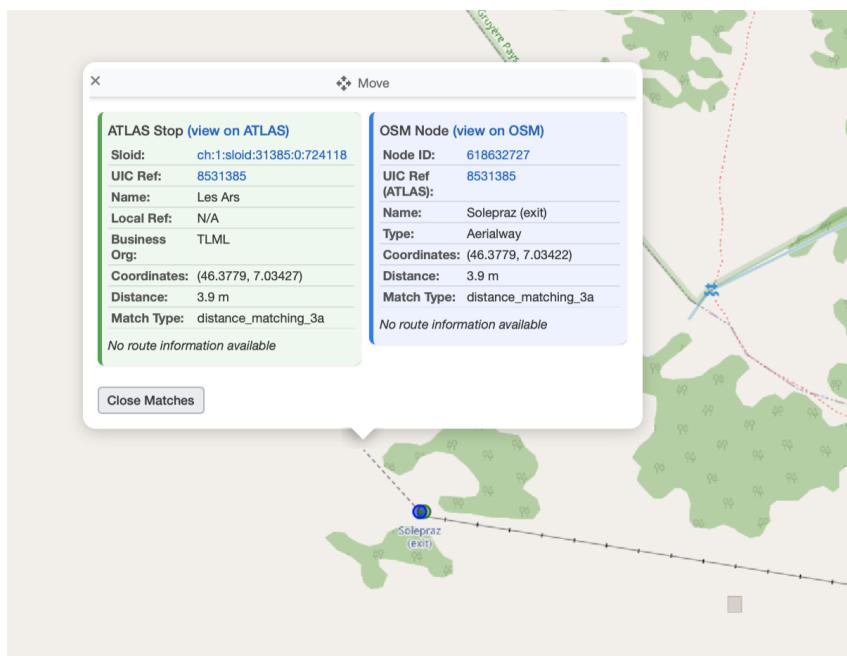


ILLUSTRATION 3.5 – Correspondance par distance étape 3 : exemple d'un arrêt isolé où un seul candidat OSM est trouvé dans le rayon de 50 mètres.

Note : La méthode de correspondance par routes, qui est la prochaine étape logique dans notre processus séquentiel, est détaillée dans le chapitre suivant en raison de sa complexité.

3.5. CONSOLIDATION POST-TRAITEMENT

Après l'exécution des méthodes principales de correspondance (exacte, par nom, par distance et par routes), le système effectue une consolidation post-traitement. Cette étape, appelée "consolidation unique par UIC", examine les entrées ATLAS restantes non appariées et recherche des nœuds OSM disponibles partageant le même identifiant UIC.

Cette consolidation est particulièrement utile dans les cas où :

- Des nœuds OSM étaient temporairement indisponibles lors de la correspondance exacte initiale
- Des conflits de priorité ont empêché certaines correspondances exactes évidentes
- Des entrées ont été filtrées lors des étapes précédentes mais redeviennent candidates valides

Le processus fonctionne de manière conservative : il ne crée une correspondance que si exactement un nœud OSM disponible correspond à l'identifiant UIC de l'entrée ATLAS, garantissant ainsi une fiabilité maximale.

Cette étape de consolidation a permis d'identifier 937 correspondances exactes supplémentaires.

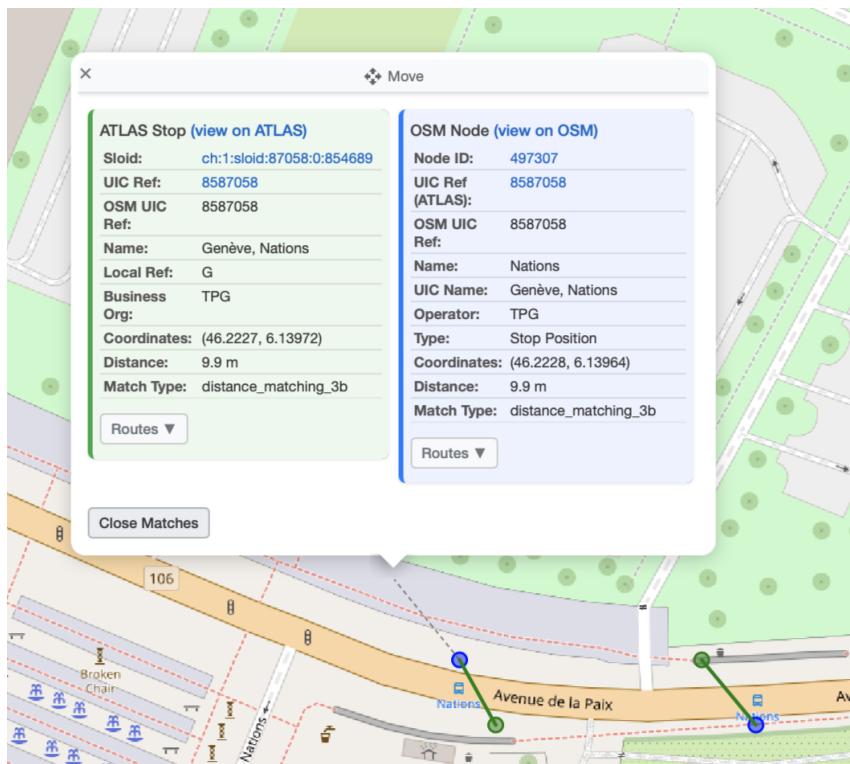


ILLUSTRATION 3.6 – Correspondance par distance étape 3b.

3.6. PROPAGATION DES DUPLICATAS

Une dernière étape consiste à propager les correspondances trouvées aux entrées ATLAS dupliquées. Lorsque plusieurs entrées ATLAS partagent les mêmes caractéristiques (numéro et désignation), et qu'une correspondance a été établie pour l'une d'entre elles, cette correspondance est étendue aux autres entrées du groupe dupliqué.

Cette propagation permet d'assurer la cohérence des correspondances et d'améliorer le taux de couverture global, particulièrement dans les grandes gares où plusieurs entrées ATLAS peuvent représenter des aspects différents d'un même arrêt physique.

La propagation des duplicates a permis d'ajouter 69 correspondances supplémentaires.

3.7. CORRESPONDANCE MANUELLE

Enfin, si les entrées n'ont pas été appariées par les méthodes précédentes, le système applique les correspondances manuelles définies préalablement par les utilisateurs et stockées de manière persistante dans la base de données.

Les correspondances manuelles sont faites depuis l'application web comme on le verra plus tard.

3.8. RÉSULTATS ACTUELS

Sur les **55 571** arrêts ATLAS considérés, nous obtenons **48 419** correspondances, comptées *en paires ATLAS–OSM* (chaque couple *slobid ATLAS–nœud OSM*). Un même arrêt pouvant être

associé à plusieurs nœuds OSM, **1 456** arrêts présentent des appariements multiples, soit **1 555** paires en sus du nombre d'arrêts distincts appariés.

Dans cette convention « par paires », la couverture atteint **87,1%** (48 419/55 571). Après déduplication par *sloïd* ATLAS, on compte **46 864** arrêts appariés, soit **84,3%** (46 864/55 571).

Méthode	Nombre	%
Correspondances exactes	21 237	43,9%
Correspondances par nom	537	1,1%
Correspondances par distance	18 618	38,5%
• Étape 1 (groupe-proximité)	15 174	31,3%
• Étape 2 (référence locale)	129	0,3%
• Étape 3a (candidat unique)	2 123	4,4%
• Étape 3b (ratio de distance)	1 192	2,5%
Correspondances par routes	7 021	14,5%
Consolidation post-traitement	937	1,9%
Propagation des duplicates	69	0,1%
Total des correspondances (paires)	48 419	
Ajustement de déduplication (appariements multiples)	-1 555	
Total des correspondances (arrêts ATLAS distincts)	46 864	

Les pourcentages par *méthode* indiquent la part des **48 419** correspondances (paires) attribuée à chaque étape et totalisent 100%. La ligne « Total des correspondances (paires) » renvoie la couverture calculée *par paires* sur **55 571** arrêts ATLAS. La ligne « Ajustement de déduplication » retranche les **1 555 appariements multiples**. Le « Total des correspondances (arrêts ATLAS distincts) » correspond alors à **84,3%** (46 864/55 571), valeur affichée dans l'application web.

Après ces étapes, 8 707 entrées ATLAS restent non appariées, et 19 115 nœuds OSM restent inutilisés. Parmi ces nœuds OSM inutilisés, 14 012 sont associés à au moins une route, 14 917 possèdent une référence UIC, et 879 ont une référence locale (`local_ref`).

Parmi les entrées ATLAS non appariées, 3 911 n'ont aucun nœud OSM dans un rayon de 50 mètres.

CHAPITRE 4 : LIGNES ET APPARIEMENT PAR LIGNES

Ce chapitre propose une visite guidée de la couche *lignes* : comment nous construisons une vue unifiée des lignes à partir de GTFS et HRDF, à quoi ressemblent les données, ce que disent les chiffres, et comment nous effectuons l'appariement basé sur les lignes entre les arrêts ATLAS et les nœuds OSM.

4.1. DES FLUX BRUTS À UN FICHIER UNIQUE DE LIGNES

a. Ce qu'est `atlas_routes_unified.csv`

Nous consolidons les *signaux de ligne* issus de deux sources dans un seul fichier tabulaire :

- **GTFS** : identifiants de ligne, noms court/long, et la direction (0/1). Les directions sont dérivées via une heuristique *premier*→*dernier* par trajet, agrégée au niveau de la ligne.
- **HRDF** : noms de lignes et chaînes de direction construites comme *première gare*→*dernière gare*, à la fois par noms et par paires de codes UIC.

Chaque ligne du CSV décrit « un signal de ligne pour un arrêt » :

TABLEAU 4.1 – Structure du fichier `atlas_routes_unified.csv`

Colonne	Signification
<code>sloid</code>	Identifiant d'arrêt ATLAS
<code>source</code>	gtfs ou hrdf
<code>evidence</code>	Méthode d'inférence (p. ex. <code>gtfs_first_last</code> , <code>hrdf_fplan</code>)
<code>as_of</code>	Date d'extraction
<code>route_id, route_id_normalized</code>	ID GTFS brut et normalisé par année
<code>route_name_short, route_name_long</code>	Noms de ligne GTFS
<code>line_name</code>	Ligne HRDF (si disponible)
<code>direction_id</code>	Direction GTFS 0/1 (chaîne)
<code>direction_name, direction_uic</code>	Chaînes premier→dernier humaines et UIC

Cette structure est produite directement par l'écriture unifiée. La **normalisation par année** supprime les suffixes saisonniers (p. ex. -j24) pour stabiliser la comparaison entre les années :

Normalisation des identifiants de ligne

```

1 import re
2 def normalize_route_id(route_id: str) -> str:
3     return re.sub(r"-j\d+", "-jXX", route_id)

```

b. Comment on le génère (vue d'ensemble)

Le processus de haut niveau (voir `get_atlas_data.py`) est le suivant :

1. Charger GTFS en flux et ne garder que les arrêts suisses (IDs commençant par 85).
2. Premier passage sur `stop_times` : pour chaque `trip_id`, collecter le premier et le dernier arrêt suisses ; joindre à `trips` et `routes` pour obtenir l'ID et les noms de ligne.
3. Construire, par ligne, les chaînes de direction « nom du premier arrêt → nom du dernier arrêt » (dédupliquées).
4. Second passage sur `stop_times` : dédupliquer (`stop_id`, `route_id`, `direction_id`).
5. Mapper `stop_id` GTFS vers `sloid` ATLAS (règle stricte puis repli sûr).
6. Parser HRDF (GLEISE_LV95, FPLAN, BAHNHOF) pour obtenir lignes et directions premier→dernier par `sloid`.
7. Écrire un unique CSV propre combinant les deux sources.

4.2. CE QUE DISENT LES DONNÉES UNIFIÉES

Les chiffres ci-dessous sont calculés avec les scripts sous `mémoire/scripts_used/chap4`.

Vue GTFS

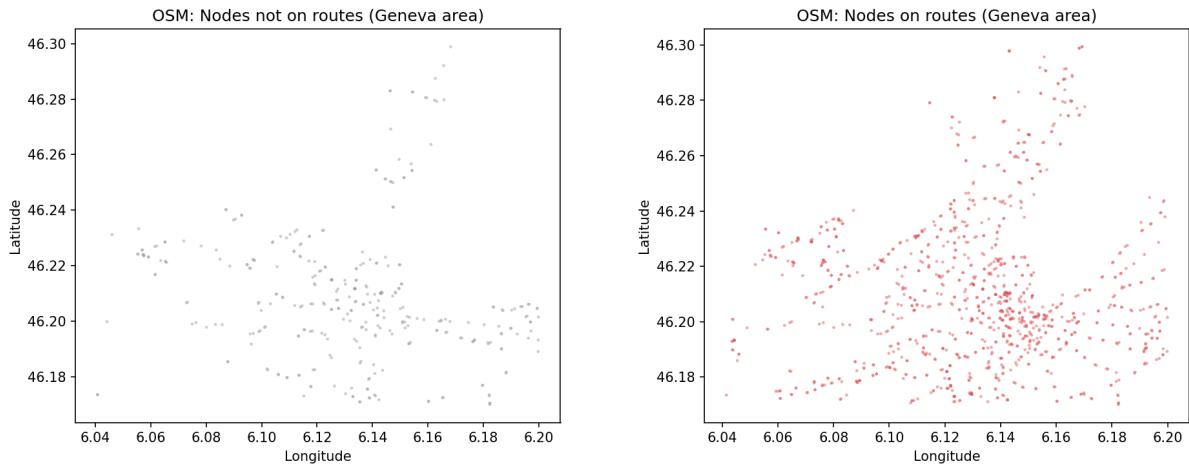
- **SLOIDs avec lignes GTFS : 34 781**
- **Nombre moyen de lignes uniques par `sloid` : 2,73** (médiane **2,00**)
- **Nombre moyen de couples (ligne, direction) par `sloid` : 4,39** (médiane **3,00**)
- **Taux de duplication pour (`sloid`, `route_norm`, `direction`) : 96,96%**
- **Même ligne+direction, plusieurs chaînes de direction** : **96,95%** des groupes présentent plus d'une chaîne premier→dernier (ce qui indique des schémas d'exploitation hétérogènes).

Vue HRDF

- **SLOIDs avec lignes HRDF : 28 757**
- **Nombre moyen de lignes uniques par `sloid` : 2,32** (médiane **2,00**)
- **Directions distinctes par (`sloid`, `line_name`) : 2,67** par UIC et **2,67** par nom (médianes **2,00**)
- **Longue traîne** : plusieurs couples (`sloid`, ligne) affichent **30–40** paires UIC premier→dernier distinctes (branches, demi-tours)

En résumé. GTFS couvre un grand nombre d'arrêts avec plusieurs directions par ligne ; HRDF confirme une forte variété de directions terminales pour certaines lignes (longue traîne).

Cela implique qu'une comparaison robuste doit gérer la multiplicité des directions, et pas seulement les identifiants de ligne.



OSM : nœuds ne faisant partie d'*aucune* relation de ligne (Genève).

OSM : nœuds participant à *au moins une* relation de ligne (Genève).

ILLUSTRATION 4.1 – Nœuds OSM hors lignes vs. sur des lignes (région de Genève).

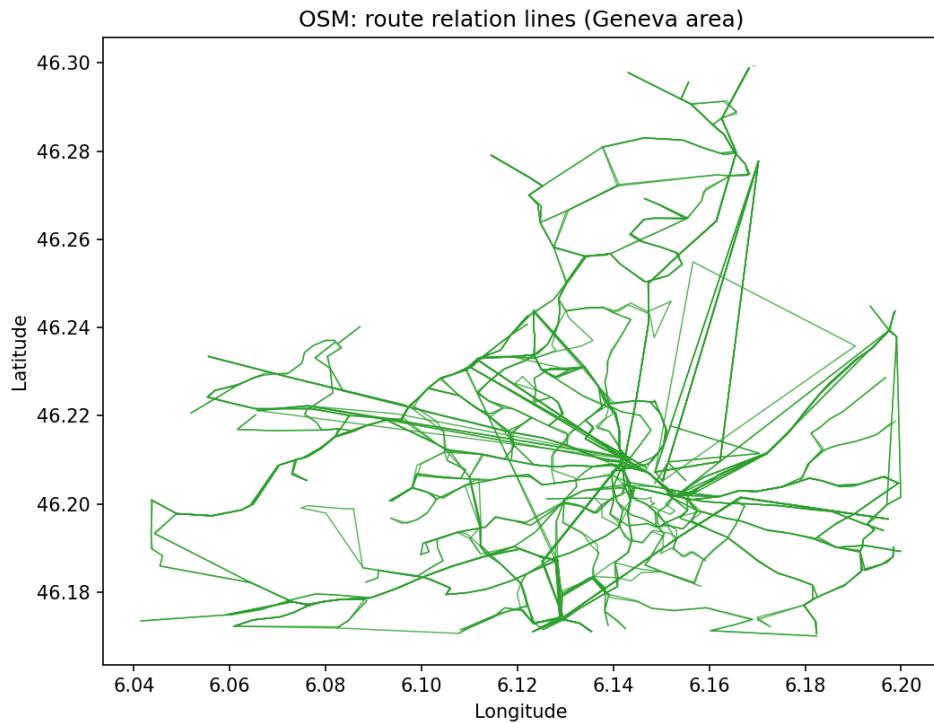


ILLUSTRATION 4.2 – OSM : tracé des lignes à partir des relations de type route (Genève). Géométrie bien structurée.

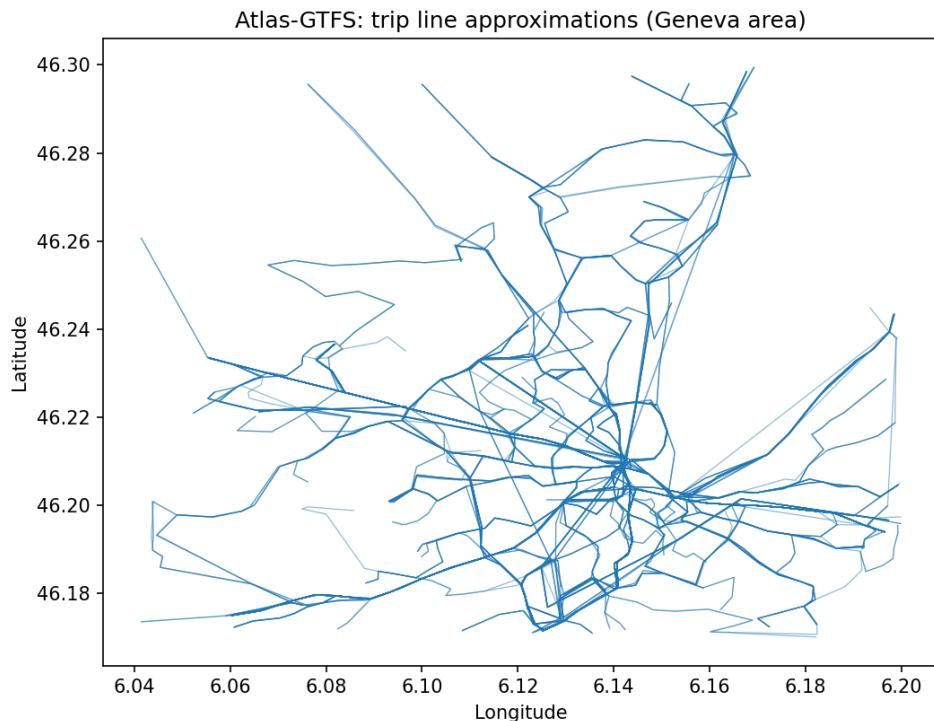


ILLUSTRATION 4.3 – Atlas-GTFS : approximation des trajets (Genève). Lignes reconstruites depuis l'ordre des arrêts — plus fragmenté.

Tracer les lignes *Atlas-GTFS* (simplifié)

```

1 stops = read_csv('stops.txt')[['stop_id','stop_lat','stop_lon']]
2 stop_times = read_csv('stop_times.txt')[['trip_id','stop_id','stop_sequence']]
3
4 # 1) Restreindre aux arrêts dans la boîte Genève
5 stops_ge = stops[in_bbox(stop_lat, stop_lon)]
6 stop_times_ge = stop_times[stop_id \in stops_ge.stop_id]
7
8 # 2) Reconstituer les séquences ordonnées par trajet
9 seq_map = {tid: tuple(g.sort_values('stop_sequence').stop_id)
10           for tid, g in stop_times_ge.groupby('trip_id') if len(g) >= 2}
11
12 # 3) Compter les séquences répétées et en échantillonner
13 seq_counts = Counter(seq_map.values())
14 chosen = choose_top_sequences(seq_counts, max_trips=500)
15
16 # 4) Projeter en coordonnées et tracer
17 for seq in chosen:
18     pts = [stops_ge.loc[id][('stop_lat','stop_lon')] for id in seq]
19     draw_polyline(filter_in_bbox(pts))

```

4.3. D'AUTRES VUES UTILES

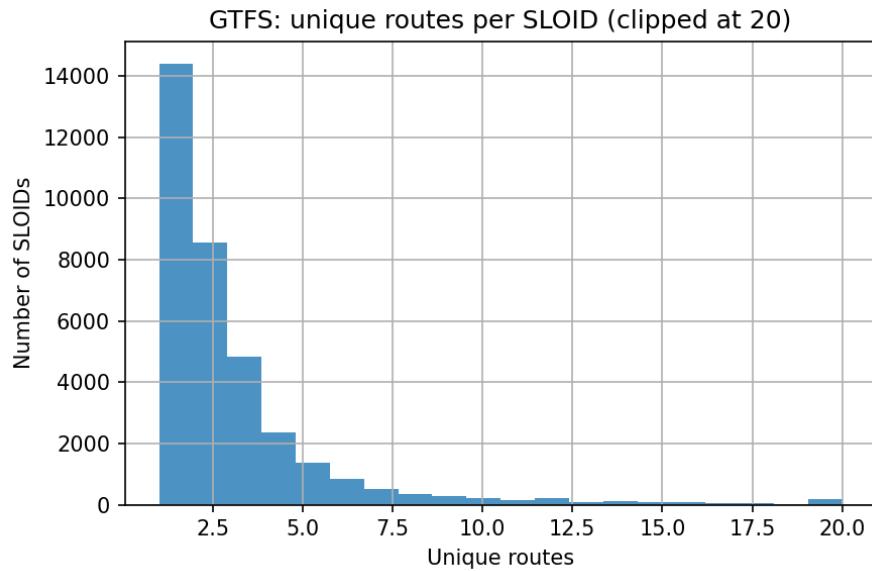


ILLUSTRATION 4.4 – Distribution du nombre de lignes GTFS uniques par sloid (coupée à 20).

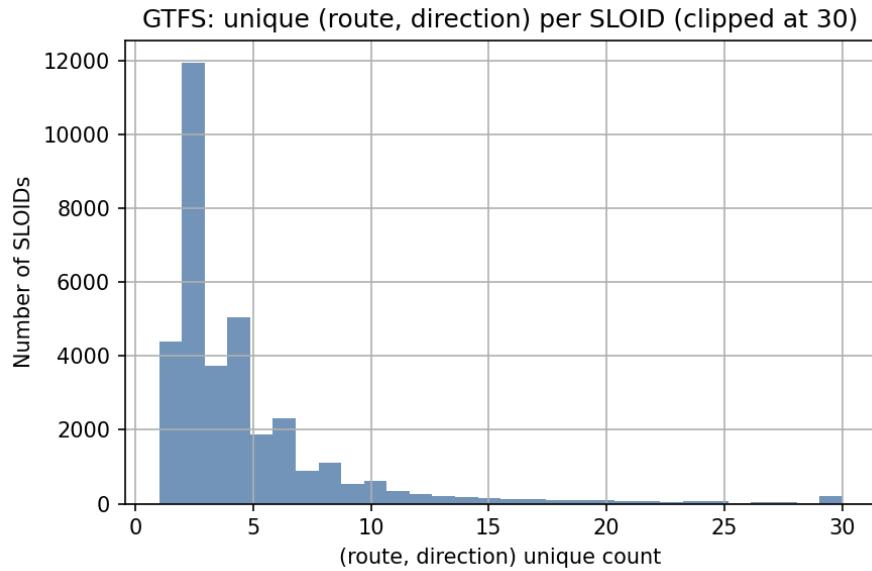


ILLUSTRATION 4.5 – Distribution du nombre de couples (ligne, direction) par sloid (coupée à 30).

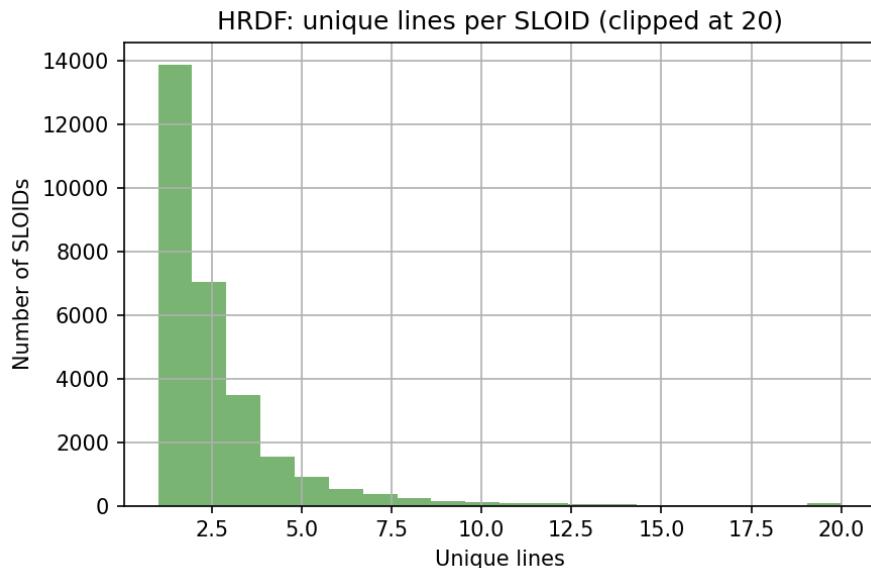


ILLUSTRATION 4.6 – Distribution du nombre de lignes HRDF uniques par sloid (coupee a 20).

- **GTFS — lignes par sloid** : moyenne **2,733**, mediane **2,0**, p10 **1**, p90 **5**, max **58**. *Lecture* : la plupart des arrêts ont 2–5 lignes, quelques hubs dépassent largement.
- **GTFS — (ligne, direction) par sloid** : moyenne **4,392**, mediane **3,0**, p10 **1**, p90 **8**, max **95**. *Lecture* : les directions doublent naturellement la diversité par rapport aux seules lignes.
- **HRDF — lignes par sloid** : moyenne **2,318**, mediane **2,0**, p10 **1**, p90 **4**, max **33**. *Lecture* : structure comparable a GTFS, avec des extremes moins frequents.

Comment obtenons-nous les *tokens* HRDF UIC ?

Rappel synthétique de notre méthode (déttaillée au Chapitre 1), qui s'appuie sur trois fichiers HRDF principaux pour extraire les informations de direction :

- **GLEISE_LV95** est d'abord utilisé pour lier chaque sloid de quai à l'UIC de sa gare et à une référence de quai locale (#ref). Cela nous permet d'identifier tous les voyages desservant un quai spécifique.
- **FPLAN** est ensuite consulté pour les voyages pertinents afin de déterminer leur direction. Nous y extrayons le premier et le dernier arrêt de chaque trajet (sous forme de codes UIC), ainsi que le nom de la ligne.
- **BAHNHOF** fournit les noms de gares correspondant aux codes UIC, ce qui nous permet de construire des chaînes de direction lisibles (par exemple, "Genève → Lausanne") ainsi que des chaînes basées sur les UIC (par exemple, "8501008 → 8501120").

Ces tokens (`line_name`, `direction_uic`) sont comparés aux chaînes UIC premier→dernier dérivées d'OSM pour appuyer l'appariement au niveau HRDF lorsque les identifiants GTFS

manquent dans OSM.

4.4. FONCTIONNEMENT DE L'APPARIEMENT PAR LIGNES

L'appariement compare les tokens de ligne connus pour un `sloid` ATLAS aux tokens dérivés des nœuds OSM à proximité (KD-tree, rayon configurable : 50 m par défaut). Les tokens sont soit **GTFS** (`route_id`, `direction_id`), avec normalisation, soit **HRDF** (`line_name`, `direction_uic`).

a. Candidats par distance

Nous tentons l'appariement en quatre paliers :

1. **P1/P2 (tokens GTFS)** : intersection non vide entre les tokens GTFS du `sloid` et ceux d'un noeud candidat.
2. **P3 (HRDF par UIC)** : présence d'une chaîne UIC premier→dernier du côté du noeud (membre d'une relation OSM) correspondant à une chaîne HRDF du `sloid`.
3. **P4 (repli par noms)** : concordance entre une chaîne *nominales* OSM premier→dernier et une chaîne unifiée (côte ATLAS).

Extrait minimaliste de la logique des tokens :

Intersection de tokens GTFS

```

1 node_tokens = set()
2 for route in node_routes:
3     rid = route.gtfs_route_id
4     did = route.direction_id or '0'
5     if rid:
6         node_tokens.add((rid, did))
7         rid_norm = normalize_route_id(rid) # '-j25' -> '-jXX'
8         if rid_norm:
9             node_tokens.add((rid_norm, did))
10
11 if gtfs_tokens & node_tokens:
12     match = ('gtfs', 'gtfs_tokens')

```

La normalisation des identifiants de ligne utilisée dans tout le système est :

Normalisation route_id

```

1 import re
2
3 def normalize_route_id(route_id: str) -> str:
4     return re.sub(r"-j\d+", "-jXX", route_id)

```

b. Paramètres

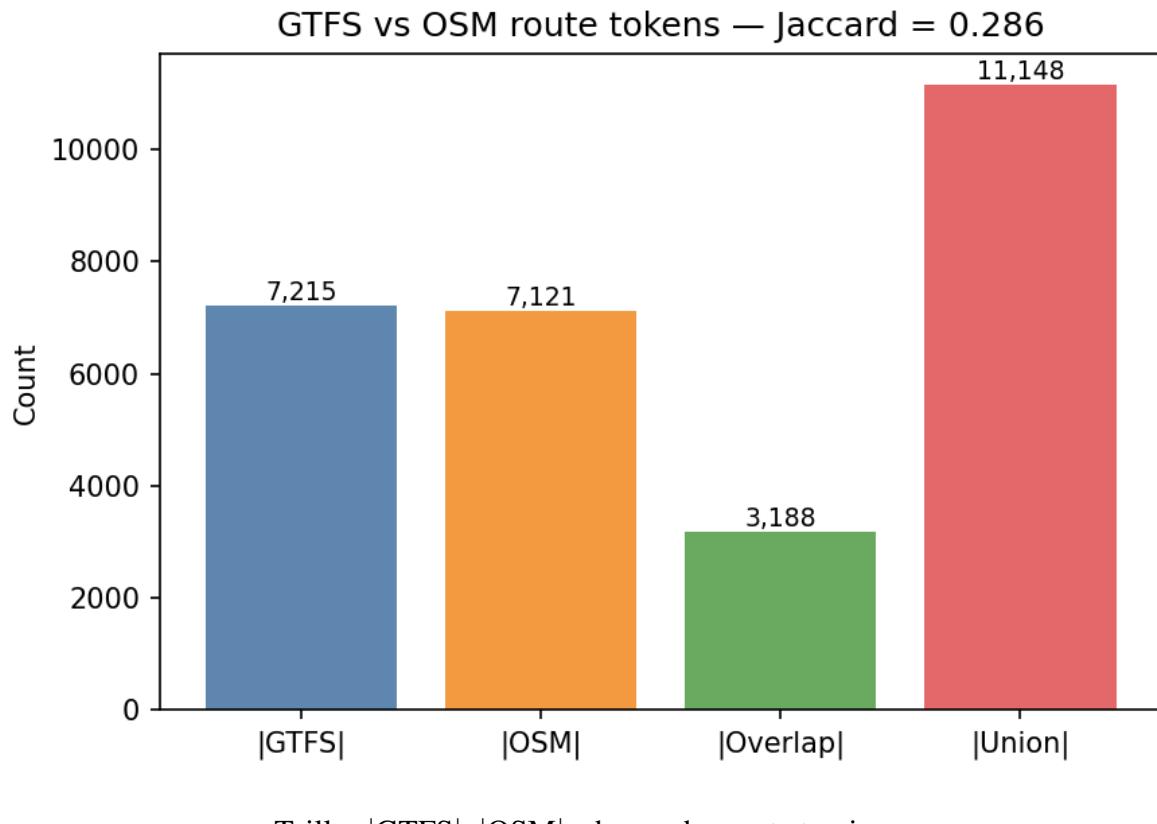
- **Rayon** : 50 m par défaut. Plus petit \Rightarrow moins de faux positifs, mais risque de manquer des arrêts légèrement décalés dans OSM.
- **Types de tokens** : activer uniquement GTFS ou inclure les paliers HRDF.
- **Normalisation** : comparer avec ou sans -jXX.

4.5. RÉSULTATS DE L'APPARIEMENT PAR LIGNES

- **Tokens GTFS** : 7 252; **tokens OSM** : 7 121; **chevauchement** : 3 200; **Jaccard** : **0,2864**.
- **Couverture par slloid (GTFS)** : moyenne **2,64**, médiane **2**, p90 **6**; au moins un token couvert pour **73,4%** des sloids.
- **Au niveau des lignes** : lignes GTFS uniques **3 839**; avec correspondance OSM **1 664** \rightarrow **43,3%** de lignes appariées.
- **Chaînes UIC premier \rightarrow dernier** : HRDF **8 818**, OSM **5 633**, chevauchement **2 935**.

Interprétation : la similarité **Jaccard** $\approx 0,29$ indique un recouvrement substantiel mais non total des tokens GTFS dans OSM; le taux d'appariement **43%** au niveau des lignes confirme une couverture utile pour un appariement de haute précision.

Illustration de la similarité de Jaccard. Rappel : $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.



a. Efficacité de l'appariement par lignes : une analyse complète

Pour quantifier l'efficacité réelle de l'appariement par lignes, nous avons mené une **analyse comparative** entre l'appariement exact seul et l'appariement par lignes seul sur l'ensemble complet des arrêts ATLAS. Cette évaluation révèle des enseignements cruciaux sur la valeur ajoutée et la complémentarité des deux méthodes.

Méthodologie. L'étude compare deux pipelines isolés : (1) appariement exact basé uniquement sur les références UIC, et (2) appariement par lignes utilisant exclusivement les tokens de lignes GTFS et HRDF. Chaque méthode opère sur l'ensemble des 56 515 arrêts ATLAS avec un rayon de 50 mètres.

Résultats quantitatifs.

- **Appariement exact** : 21 250 paires arrêt↔nœud
- **Appariement par lignes** : 29 582 paires arrêt↔nœud
- **Intersection** : 6 971 paires communes aux deux méthodes
- **Similarité Jaccard** : 0,159 — faible recouvrement, forte complémentarité
- **Précision de l'appariement par lignes** : 23,6% des correspondances de lignes sont confirmées par l'appariement exact

- **Valeur ajoutée** : **40,0%** des arrêts ATLAS obtiennent de nouvelles correspondances uniquement via les lignes

Qualité spatiale. L'appariement par lignes maintient une précision spatiale élevée : distance moyenne de **11,9 m**, médiane **8,2 m**, et **86%** des correspondances à moins de 25 mètres. Cette qualité spatiale démontre que les tokens de ligne, bien qu'indirects, identifient des noeuds OSM géographiquement cohérents.

L'analyse révèle une **complémentarité remarquable** : l'appariement par lignes trouve **22 611** nouvelles paires que l'appariement exact ne détecte pas, soit un **facteur de complémentarité de 1,58**. Cela signifie que pour chaque correspondance manquée par l'appariement exact, l'appariement par lignes en propose environ 1,6 nouvelles.

Cette performance s'explique par la richesse des données de lignes (**34 781** arrêts ATLAS avec données GTFS, **28 757** avec HRDF) et la couverture étendue d'OSM (**51 286** noeuds avec informations de lignes). L'appariement par lignes exploite ainsi des signaux de transport public que l'appariement exact, limité aux références UIC explicites, ne peut capturer.

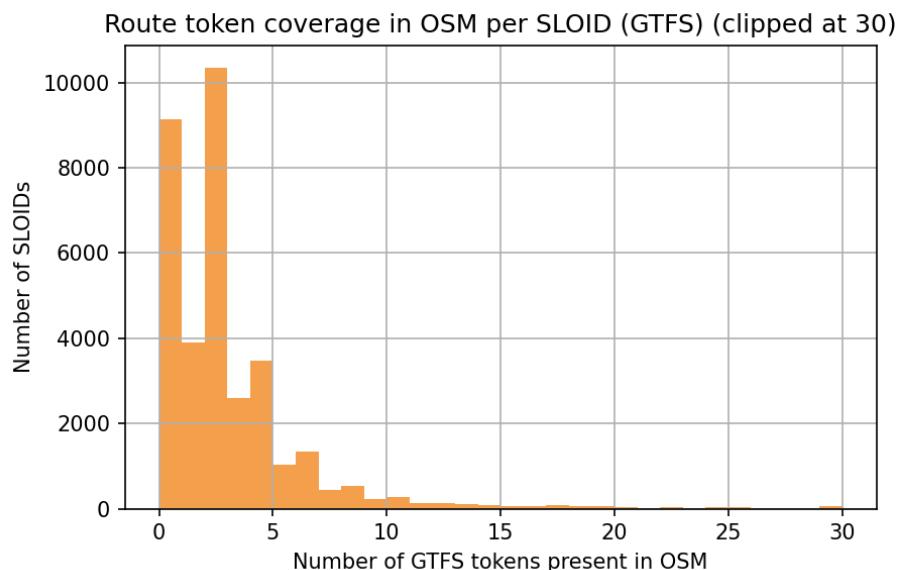


ILLUSTRATION 4.7 – Nombre de tokens GTFS (*ligne,direction*) par *sloid* retrouvés dans OSM (coupée à 30).

4.6. BILAN ET PERSPECTIVES

L'appariement par lignes ajoute un signal fort et indépendant qui complète les méthodes exactes, nominales et par distance. L'analyse d'efficacité démontre sa valeur ajoutée substantielle (**40%** de nouveaux appariements) tout en maintenant une qualité spatiale élevée (86% des correspondances sous 25 m).

CHAPITRE 5 : ANALYSE DES RÉSULTATS

Ce chapitre synthétise la qualité des correspondances ATLAS ↔ OSM. Nous mettons l’accent sur : lecture par méthode, variations par opérateur, couverture par tranches de distance, et cas extrêmes.

5.1. LECTURE GLOBALE PAR MÉTHODE

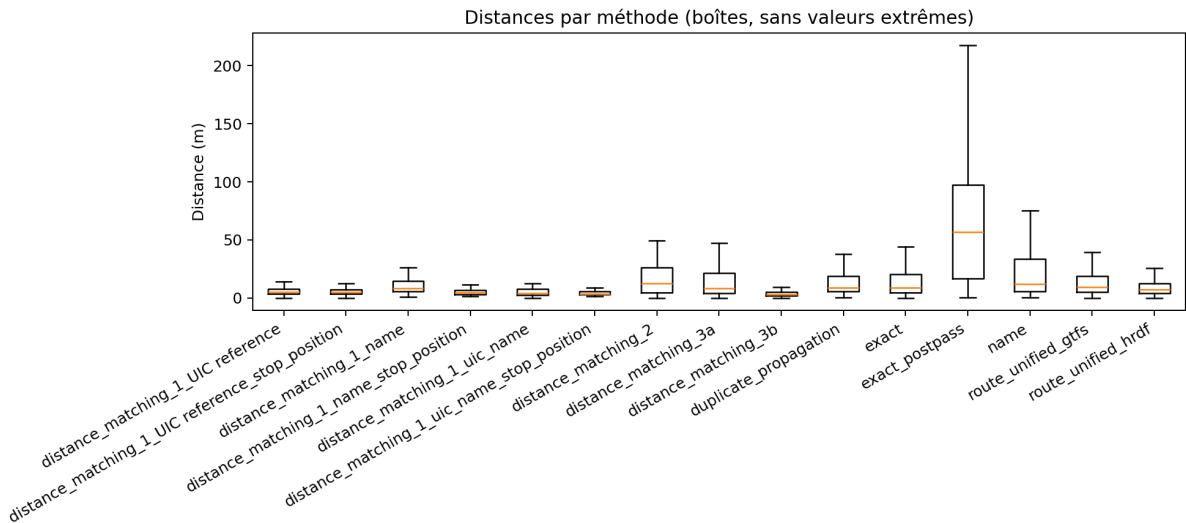


ILLUSTRATION 5.1 – Boîtes à moustaches par méthode (sans valeurs extrêmes) — médianes et dispersion.

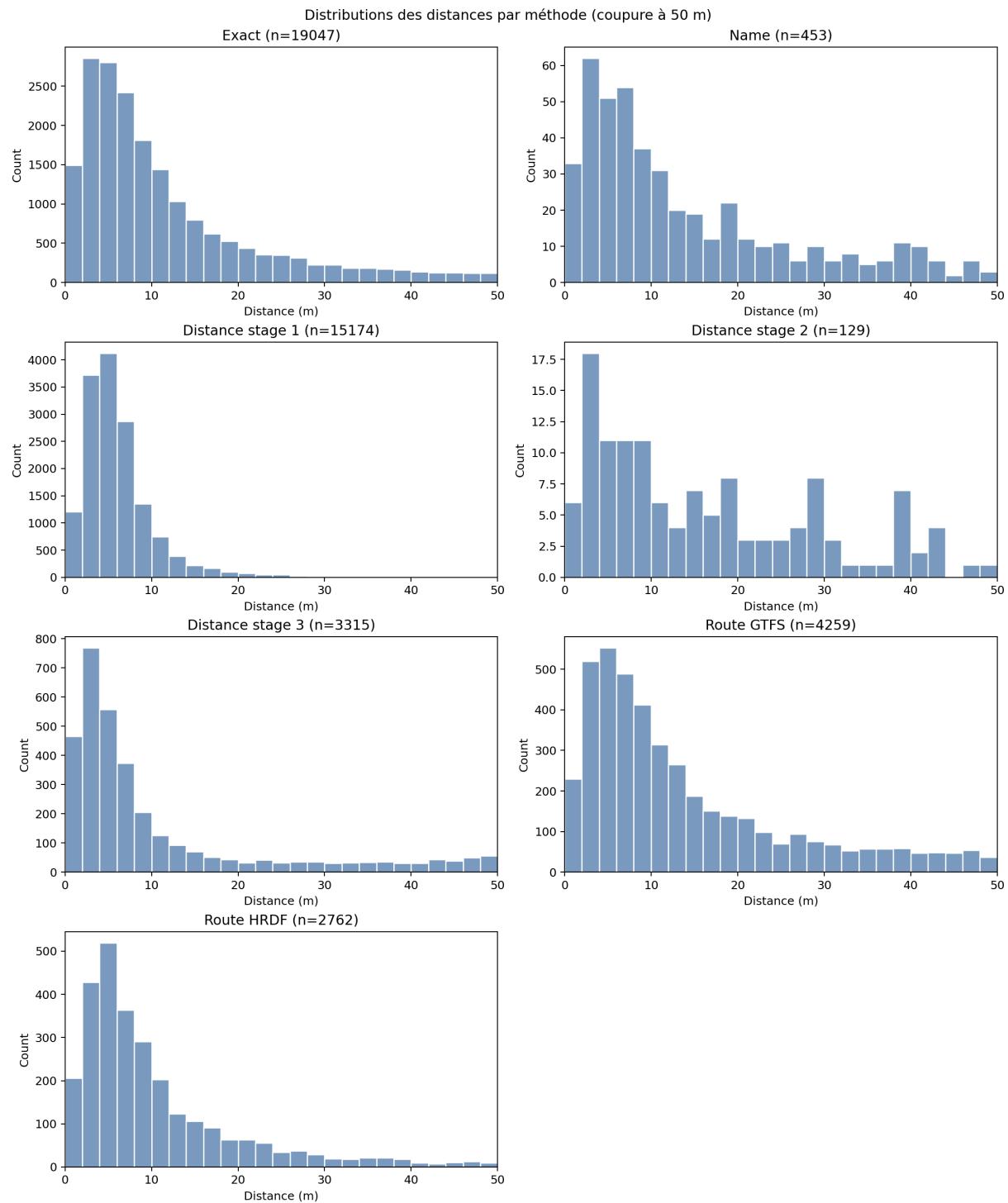


ILLUSTRATION 5.2 – Histogrammes des distances coupées à 50 m par méthode : Exact, Name, Distance (stages 1–3), Route GTFS, Route HRDF. Note : sauf Exact et Name, les méthodes de distance et de routes sont par construction limitées à 50 m.

On observe que les distributions des distances sont très proches entre les méthodes, avec des médianes et des dispersions similaires. C'est un bon signe : les approches Exact, Name, Distance (stages 1–3) et Routes (GTFS/HRDF) convergent vers un niveau de précision comparable, sans biais systématique marqué.

5.2. PAR OPÉRATEUR : OÙ EST-CE LE PLUS PRÉCIS ?

Nous relierons chaque arrêt à son opérateur ATLAS et observons les distances des arrêts *appariés*. Ci-dessous, la dispersion pour les opérateurs les plus représentés.

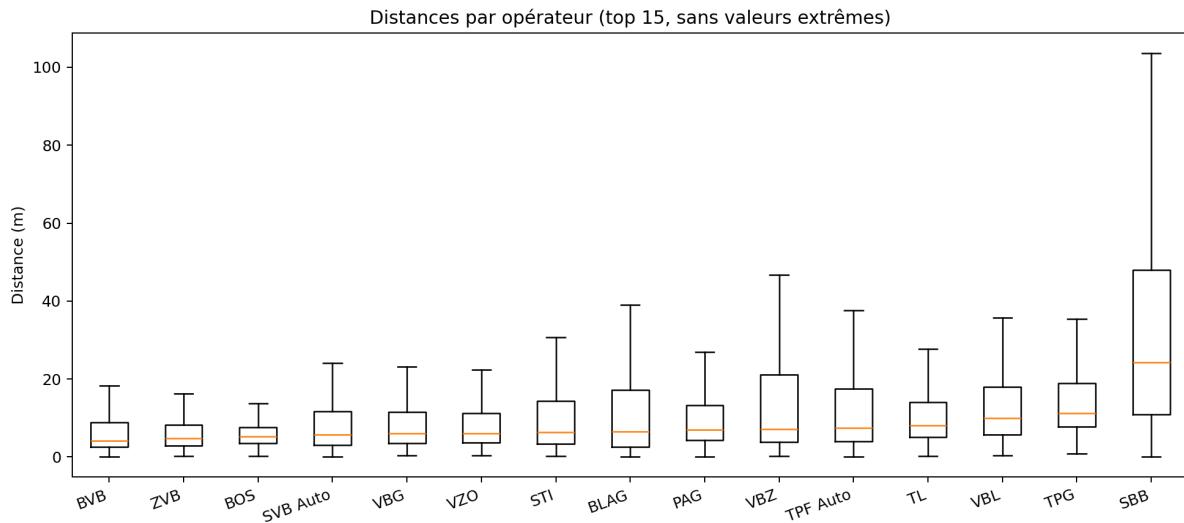


ILLUSTRATION 5.3 – Dispersion des distances par opérateur (top 15 par effectif).

5.3. RÉPARTITION PAR TRANCHES DE DISTANCE

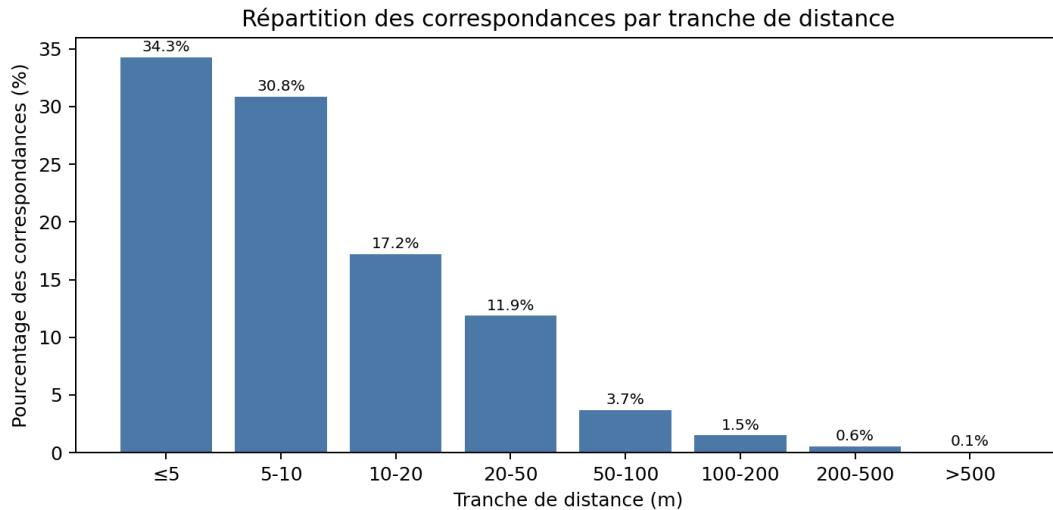


ILLUSTRATION 5.4 – Part des correspondances sous 5 m, 10 m, 20 m, etc.

5.4. CAS EXTRÊMES : TOP 10 PLUS GRANDES DISTANCES

Table des 10 plus grandes distances parmi les correspondances, avec désignations officielles ATLAS/OSM, méthode d'appariement, et UIC (ATLAS/OSM).

ATLAS (officiel)	OSM (officiel)	Méthode	UIC OSM	ATLAS / OSM	Distance (m)
Bex, Les Luisances	Les Plans-sur-Bex, Pont de Nant	exact_post- pass	8570643 8570643	/	6650.2
Villars-sur-Ollon, La Roche	Villars-sur-Ollon, La Roche	exact	8501707 8501707	/	3226.7
Villars-sur-Ollon, La Roche	Villars-sur-Ollon, La Roche	exact	8501707 8501707	/	3226.6
Col-de-Chassoure (Tortin)	Col-de-Chassoure (Tortin)	exact	8530100 8530100	/	2487.7
Cabane-des-Violettes	Cabane-des-Violettes	exact	8530169 8530169	/	1879.0
Naraus	Foppa (Naraus)	name	8530548 8530879	/	1525.3
Espel (Talst.Stöfelibahn)	Espel (Talst.Stöfelibahn)	exact	8500277 8500277	/	1451.9
Piz Mundaun	Piz Mundaun	exact	8530543 8530543	/	1413.3
Faido	Faido	name	8505204 8505204	/	892.9
Diga di Luzzzone/Ristorante	Diga di Luzzzone/Ristorante	exact	8588141 8588141	/	846.8

TABLEAU 5.1 – Top 10 des plus grandes distances — repérage des cas à investiguer.

CHAPITRE 6 : DÉTECTION DES PROBLÈMES

Ce chapitre présente un aperçu de la détection des problèmes après l'appariement des arrêts ATLAS avec les nœuds OSM. Nous y décrivons les types de problèmes, expliquons les logiques de priorisation et concluons par une réflexion sur des pistes d'amélioration.

6.1. TYPES DE PROBLÈMES DÉTECTÉS

Notre pipeline signale les problèmes au niveau des arrêts consolidés (`stops`) et les enregistre dans une table normalisée nommée `problems`. Quatre catégories de problèmes sont actuellement gérées :

- **Problèmes de distance** — une paire ATLAS–OSM appariée est trop éloignée (en mètres).
- **Problèmes de non-appariement** — un arrêt existe dans une source de données (ATLAS ou OSM) sans correspondant plausible à proximité.
- **Problèmes d'attributs** — des conflits de métadonnées sont détectés sur des paires appariées (opérateur, nom officiel, référence locale, UIC).
- **Doublons (informationnels)** — des SLOIDs dupliqués côté ATLAS ou des nœuds OSM dupliqués pour la même plateforme (`uic_ref`, `local_ref`) sont identifiés ; ces cas sont utiles pour la révision mais ne constituent pas des contradictions en soi.

Chaque problème détecté est caractérisé par un *type*, une *solution* optionnelle (pouvant être renseignée ultérieurement via des « solutions persistantes »), et une **priorité** numérique (de P1, la plus élevée, à P3, la plus basse) afin de guider l'effort de correction.

6.2. NÉCESSITÉ DE LA PRIORISATION

Les problèmes détectés sont nombreux et de criticité variable. Par exemple, un écart de 120 mètres pour un arrêt situé dans un pôle d'échanges majeur est plus problématique qu'une légère différence dans le nom d'un opérateur. Les règles de priorisation sont donc conçues pour refléter cette réalité, afin que l'interface web mette en évidence les anomalies les plus importantes en premier.

6.3. MÉTHODE D'ATTRIBUTION DES PRIORITÉS

Nous calculons les priorités par catégorie de problèmes à l'aide de règles simples. Les règles ci-dessous décrivent la logique appliquée, sans entrer dans les détails d'implémentation.

a. Priorisation des problèmes de distance

Les critères sont moins stricts pour les arrêts du réseau ferroviaire opérés par les CFF. Les priorités sont attribuées comme suit (pour les paires `matched`) :

- **P1** — opérateur non CFF et distance strictement supérieure à 80 m.

- **P2** — opérateur non CFF et distance entre 25 m et 80 m (inclus à 80 m).
- **P3** — opérateur CFF et distance strictement supérieure à 25 m ; ou, quel que soit l'opérateur, distance entre 15 m et 25 m (inclus à 25 m).
- **Aucun problème** — distance inférieure ou égale à 15 m, ou distance indisponible.

Ces règles s'appliquent aux paires appariées (*matched*). Les résolutions de problèmes peuvent être rendues **persistantes** afin d'être réappliquées lors des imports futurs (voir § 6.6).

b. Priorisation des non-appariements

Principe : un arrêt isolé, sans correspondant à proximité, est considéré comme un problème prioritaire. Pour guider la revue, nous utilisons les paliers suivants, fondés sur la distance au plus proche voisin et, quand c'est pertinent, sur les identifiants (UIC, quais) :

Entrée ATLAS uniquement

- **P1** — aucun nœud OSM plausible à proximité (aucun voisin ou voisin au-delà de 80 m).
- **P2** — plus proche nœud OSM au-delà de 50 m ; ou incohérences de comptages UIC/quais à proximité.
- **P3** — cas restants (un voisin plausible existe à moins de 50 m, sans autre alerte forte).

Nœud OSM uniquement

- **P1** — aucun arrêt ATLAS partageant le même UIC dans le voisinage.
- **P2** — plus proche arrêt ATLAS au-delà de 50 m ; ou incohérences de comptages UIC/quais.
- **P3** — cas restants (proximité raisonnable sans indice d'incohérence).

Les distances au plus proche voisin sont calculées à l'aide d'un KD-tree sur des coordonnées sur la sphère unité, une méthode rapide et numériquement stable. Le rayon d'isolement utilisé est de **50 m** : la distance linéaire d est convertie en angle en radians par $r = 2 \sin\left(\frac{d}{2R_\oplus}\right)$, puis on interroge le KD-tree pour vérifier la présence d'au moins un voisin dans ce rayon. Un arrêt est dit *isolé* s'il n'y a aucun voisin.

c. Priorisation des problèmes d'attributs

Principe : certains champs (UIC, noms officiels) sont des identifiants critiques et ont donc plus de poids que des champs descriptifs ou opérationnels. Les priorités sont attribuées comme suit (uniquement lorsque les valeurs existent des deux côtés) :

- **P1 — UIC différent** (correspondance exacte attendue) *ou nom officiel différent* (comparaison insensible à la casse).

- **P2** — local_ref différent (insensible à la casse).
- **P3** — opérateur différent (insensible à la casse).
- **Aucun problème** — aucune divergence détectée parmi ces champs.

6.4. INTERFACE DE L'APPLICATION WEB

L'interface propose des filtres simples pour permettre un tri rapide :

- **Filtrer par type de problème** : distance, unmatched, attributes, duplicates.
- **Filtrer par priorité** : P1, P2, P3.
- **Trier** : p. ex. distance la plus grande d'abord, ou par ordre alphabétique du nom d'arrêt.
Plus de détails dans le chapitre 9.

6.5. STATISTIQUES

TABLEAU 6.1 – Indicateurs clés de la dernière exécution

Métrique	Nombre
Total des arrêts importés	75 359
Problèmes de distance	10 124
Problèmes de non-appariement	30 396
Problèmes d'attributs	14 896
Entrées avec problèmes multiples	5 204
Entrées saines (aucun problème)	22 464
Entrées avec au moins un problème (dérivé)	52 895

TABLEAU 6.2 – Répartition des problèmes par type et priorité

Type de problème	P1	P2	P3	Total
Problèmes de distance	1 171	4 179	4 774	10 124
Non-appariements	8 192	17 647	4 557	30 396
Problèmes d'attributs	7 045	197	7 654	14 896

TABLEAU 6.3 – Répartition détaillée des non-appariements

Catégorie	Nombre
ATLAS uniquement	8 416
OSM uniquement	21 980
Total non-appariements	30 396

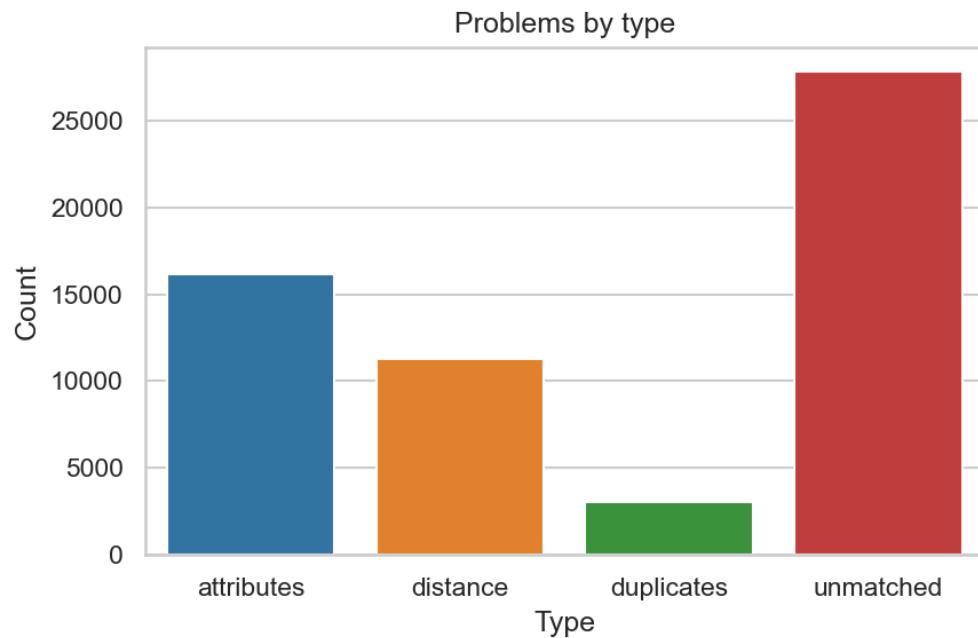


ILLUSTRATION 6.1 – Répartition des problèmes par type.

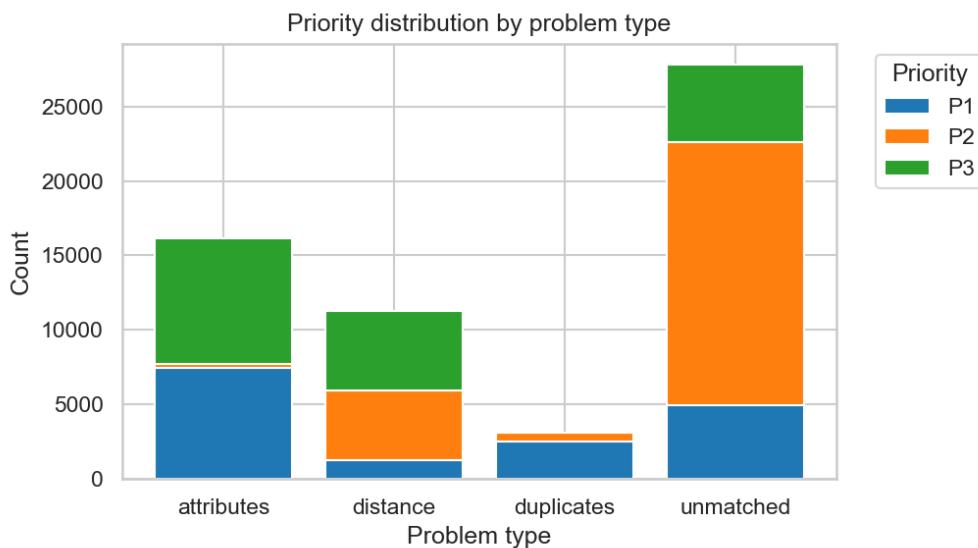


ILLUSTRATION 6.2 – Distribution des priorités (P1/P2/P3) par type de problème.

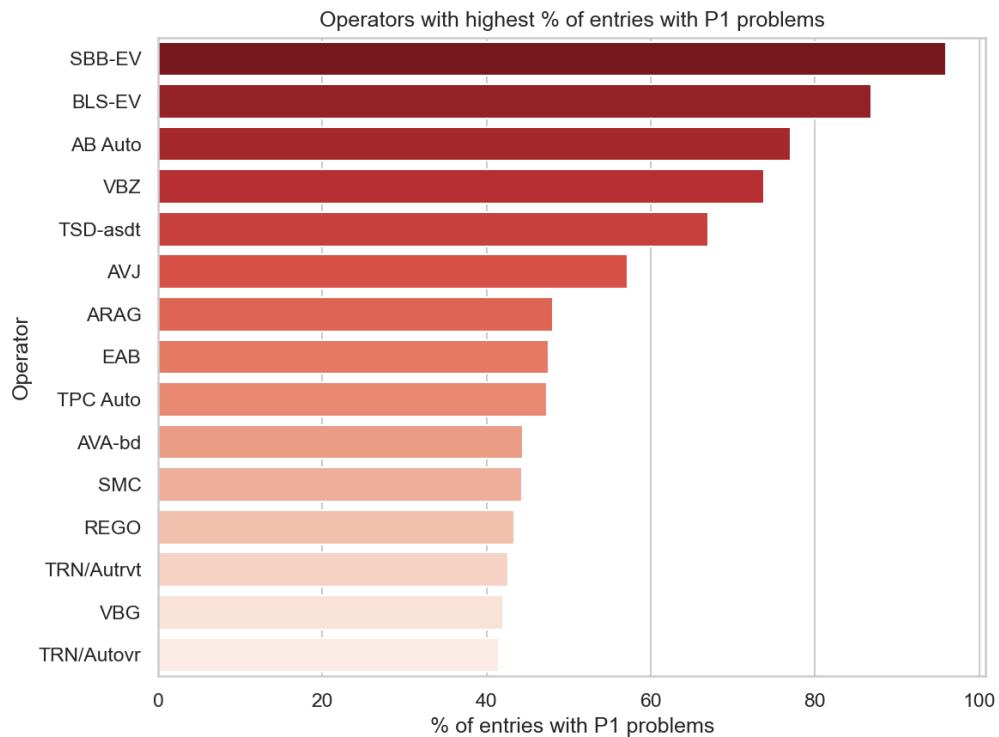


ILLUSTRATION 6.3 – Opérateurs avec le pourcentage le plus élevé d’entrées présentant des problèmes de priorité P1.

6.6. PERSISTANCE ET FLUX DE TRAVAIL DE RÉVISION

Deux mécanismes rendent la revue des problèmes efficace dans le temps :

- **Solutions persistantes** — une fois un problème résolu manuellement (p. ex. marqué *manual* pour un non-appariement), la décision est conservée et réappliquée lors des imports suivants.
- **Notes par côté** — les réviseurs peuvent ajouter des notes ATLAS/OSM qui persistent et s’affichent auprès de l’arrêt.

L’application de ces données persistantes s’effectue à la fin du processus d’import : pour chaque solution enregistrée, les arrêts concernés sont retrouvés, le problème ciblé est mis à jour avec la solution et marqué comme *persistant* afin d’être réappliqué automatiquement lors des prochains imports.

CHAPITRE 7 : BASE DE DONNÉES ET DONNÉES PERSISTANTES

POURQUOI UN CHAPITRE SUR LA BASE DE DONNÉES ?

Cette application est construite autour d'un *graphe spatial* de points d'arrêt. Le rendu cartographique en temps réel, l'analyse des problèmes et la consolidation des correspondances s'appuient sur un schéma de base de données optimisé pour la lecture, avec une logique d'*ingestion* explicite.

Ce chapitre rend cette mécanique visible, avec des extraits de code et schémas.

Le chapitre `chap8.tex` plongera ensuite dans le backend (API, endpoints, sécurité, pagination, etc.), en s'appuyant sur les éléments que nous posons ici.

Essentiels à retenir

- **Ligne auto-suffisante pour la carte** : une ligne `stops` suffit pour dessiner un marqueur (ATLAS ou OSM) \Rightarrow `pas de JOIN avec scroll`. **Fenêtre cartographique SARGable** : `indexatlas_lat/lon&osm_lat/lon` \Rightarrow `filtrage par bbox et RSSlectif`.
- **Détails séparés** : `atlas_stops` et `osm_nodes` servent les popups à la demande (*lazy*).
- **Persistance durable** : `persistent_data` stocke solutions/notes qui « survivent » aux ré-imports et sont réappliquées.
- **Séparation Auth** : schéma `auth_db` isolé via SQLAlchemy binds (sécurité, gouvernance des droits ; détail au Chap. 10).

7.1. IMPORTER LES DONNÉES : LE RÔLE DE `import_data_db.py`

L'import est orchestré par le script `import_data_db.py`. Il nettoie, croise des sources, calcule des priorités de problèmes, et construit des artefacts prêts pour l'interface.

Vue d'ensemble

Pipeline d'import des données

```
1 build_route_direction_mapping() # cartographie GTFS/OSM/HRDF -> (route, direction)
                                # <-> (noeuds, SLOIDs)
2 load_route_data()           # routes par SLOID (ATLAS) et par node_id (OSM)
3 load_unified_route_data()  # vue unifiée (gtfs/hrdf) par SLOID
4 import_to_database(...)    # insère stops, détails ATLAS/OSM, routes, problèmes
5 apply_persistent_solutions() # réapplique les solutions/notes persistantes
```

L'import crée des *stops* de trois types : `matched` (paires ATLAS–OSM), `unmatched` (AT-

LAS isolé), osm (OSM isolé). Ces valeurs de `stop_type` sont appliquées par logique métier lors de l'import et peuvent être modifiées via match manuel. Les détails riches (ex : opérateur ATLAS, tags OSM, routes) sont stockés dans des tables dédiées.

Pour la ré-application des données persistantes, le script balaye la table `persistent_data` et met à jour les nouveaux enregistrements :

Ré-application des solutions persistantes

```

1 # Extrait de import_data_db.py -> apply_persistent_solutions()
2 persistent_solutions = session.query(PersistentData)\n
3     .filter(PersistentData.note_type.is_(None)).all()\n
4\n
5 for ps in persistent_solutions:\n
6     matching_stops = session.query(Stop)\n
7         .filter((Stop.sloid == ps.sloid) | (Stop.osm_node_id == ps.osm_node_id))\n
8         .all()\n
9\n
10    for stop in matching_stops:\n
11        problem = session.query(Problem).filter(\n
12            Problem.stop_id == stop.id,\n
13            Problem.problem_type == ps.problem_type\n
14        ).first()\n
15\n
16        if problem:\n
17            problem.solution = ps.solution\n
18            problem.is_persistent = True

```

7.2. SCHÉMA LOGIQUE : LES TABLES QUI COMPTENT

Le schéma applicatif est défini dans `backend/models.py` et utilise l'ORM SQLAlchemy¹² pour interagir avec la base de données. Voici les entités principales :

stops Table centrale pour le rendu cartographique. Colonnes clefs : `sloid`, `stop_type`, `match_type`, coordonnées (`atlas_lat/lon` et `osm_lat/lon`), `distance_m`, `osm_node_type`, `atlas_duplicate_sloid`.

atlas_stops Détails ATLAS par `sloid` : désignation, opérateur, routes unifiées, notes persistantes.

osm_nodes Détails OSM par `osm_node_id` : tags de transport, opérateur, routes OSM, notes.

problems Détections automatiques (`distance`, `unmatched`, `attributes`, `duplicates`), solution éventuelle, `priority` et traçabilité auteur.

12. SQLAlchemy. SQLAlchemy Documentation [en ligne]. Disponible sur : <https://www.sqlalchemy.org/> (consulté le 2025-08-04).

persistent_data Stockage des solutions et notes destinées à survivre aux ré-imports.

routes_and_directions Consolidation GTFS/HRDF/OSM avec clés multiples : (route_id, direction_id) pour GTFS/OSM et (line_name, direction_uic) pour HRDF, plus route_id_normalized pour unification.

Note sur l'authentification Un schéma séparé (auth_db) stocke les utilisateurs (users) et les événements (auth_events) via un bind SQLAlchemy distinct (`_bind_key_='auth'`). Il est volontairement isolé du schéma métier ; nous le détaillons au Chap. 10.

Diagramme conceptuel

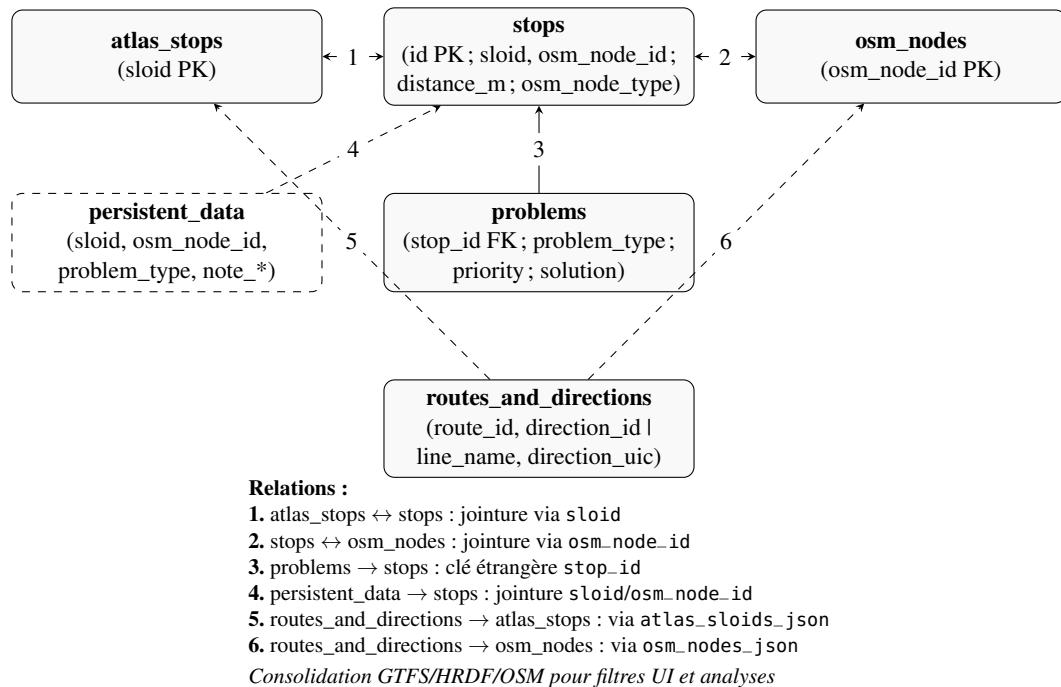


ILLUSTRATION 7.1 – Schéma relationnel des tables principales (liens via champs de jointure ; FK explicite seulement pour `problems.stop_id`).

Remarque : les liens « stops → détails » sont réalisés par *jointures explicites* (`sloid`, `osm_node_id`) plutôt que des clés étrangères rigides. Ce choix facilite l'ingestion et limite les verrouillages lors des rafraîchissements, tout en gardant des *indexes* ciblés pour les requêtes critiques. L'ORM SQLAlchemy définit des relations `lazy='joined'` sur ces champs pour optimiser les accès.

Index utiles pour la carte

- `idx_atlas_lat_lon` et `idx_osm_lat_lon` pour filtrer vite par fenêtre cartographique.
- `idx_stop_type_match_type` pour les filtres dynamiques.

- `idx_distance_m` pour les tris par distance.
- Sur `routes_and_directions` : `idx_osm_route_direction`, `idx_atlas_route_direction`, `idx_atlas_line_direction_uic`, et `idx_source` (colonne source pour différencier GTFS/HRDF). Le champ `route_id_normalized` sert à l'unification cross-sources.

7.3. PROS ET CONS DU SCHÉMA VIS-À-VIS DU RENDU CARTOGRAPHIQUE

Points forts

- **Lecture optimisée** : une ligne de `stops` suffit pour dessiner un marqueur (ATLAS ou OSM) sans JOIN.
- **SARGable viewport** : la requête de fenêtre cartographique est *selective* grâce aux index lat/lon des deux mondes (extrait d'API ci-dessous).
- **Détails séparés** : les tables `atlas_stops` et `osm_nodes` chargent les popups à la demande (lazy) sans gonfler la ligne `stops`.
- **Routes consolidées** : la table `routes_and_directions` alimente les filtres par ligne et direction côté UI.

Compromis

- **Intégrité logique** : l'absence de FK strictes suppose une discipline d'import (gérée par `import_data_db.py`).
- **Duplication contrôlée** : certaines valeurs (ex : `distance_m`) sont redondantes par design pour éviter des calculs à la volée.
- **Évolution des tags OSM** : les champs `osm_*` sont *snapshotnés* ; toute évolution nécessite un nouvel import.

La requête de fenêtre

Filtrage géographique optimisé — backend/blueprints/data.py

```
1 # Requête SARGable pour l'endpoint /api/data
2 viewport_sargable = or_()
3     # Points ATLAS dans la fenêtre
4     and_(Stop.atlas_lat.between(min_lat, max_lat),
5         Stop.atlas_lon.between(min_lon, max_lon)),
6
7     # Points OSM seuls dans la fenêtre
8     and_(Stop.atlas_lat.is_(None), Stop.atlas_lon.is_(None),
9         Stop.osm_lat.between(min_lat, max_lat),
10        Stop.osm_lon.between(min_lon, max_lon))
```

```

11 )
12
13 query = query.filter(viewport_sargable)

```

7.4. DONNÉES PERSISTANTES : COMMENT ELLES SURVIVENT AUX RÉ- IMPORTS

Le mécanisme de persistance se trouve à deux endroits : (i) l'API de gestion (`/api/make_solution_persistent`, `/api/save_note/...`), (ii) l'étape `apply_persistent_solutions()` de l'import.

Côté base

La table `persistent_data` stocke :

- des **solutions** par triplet (`slloid`, `osm_node_id`, `problem_type`) ;
- des **notes** persistantes côté ATLAS (`note_type = 'atlas'`) ou OSM ('osm').

Côté web (UI)

Dans l'interface « Problèmes », l'utilisateur peut :

- résoudre un problème, puis *rendre la solution persistante* (bouton dédié) ;
- saisir une note côté ATLAS ou OSM et la marquer persistante ;
- effectuer un *match manuel* entre deux entrées (ATLAS ↔ OSM) et l'enregistrer de manière durable.

Nous ajouterons des captures d'écran de l'interface dans une version ultérieure du manuscrit.

Un mini-exemple côté navigateur

Match manuel persistant — interface web

```

1 // Extrait simplifié de la logique JS de match manuel
2 $.ajax({
3     url: '/api/manual_match',
4     method: 'POST',
5     contentType: 'application/json',
6     data: JSON.stringify({
7         atlas_stop_id: atlasId,
8         osm_stop_id: osmId,
9         make_persistent: true
10    }),
11    success: function(response) {

```

```

12     console.log('Match persistant créé:', response);
13 }
14 });

```

Lors du prochain import, la solution manuelle réapparaîtra *sans effort* grâce à `apply_persistent_solutions()`.

7.5. PERFORMANCE : POURQUOI ÇA DÉFILE VITE SUR LA CARTE

L'expérience utilisateur fluide de la carte interactive repose sur deux ingrédients architecturaux clés :

Ligne auto-suffisante pour le rendu La table `stops` contient toutes les informations nécessaires au rendu d'un marqueur. Un *viewport query* n'a pas besoin de joindre des tables lourdes : la position et la nature du marqueur (`osm_node_type`) sont « en main ».

Index spatiaux ciblés Les clauses `BETWEEN` sur les coordonnées (`atlas_lat/lon, osm_lat/lon`) exploitent des index dédiés, permettant une sélection très rapide par fenêtre géographique.

Chargement différé des détails Les informations riches (opérateur, routes, notes) sont chargées *au clic* pour construire les popups — ce qui évite de surcharger la phase de *fetch* initial. Cette stratégie de *lazy loading* maintient des temps de réponse constants même avec des milliers de points.

Optimisations côté client Les marqueurs qui se superposent géographiquement sont « décalés » de quelques pixels afin de rester individuellement cliquables, sans impact sur les performances de rendu.

CHAPITRE 8 : BACKEND

Ce chapitre présente l'architecture backend qui alimente l'application : structure par *blueprints* Flask¹³, sérialisation, requêtage optimisé, endpoints majeurs (données, recherche, statistiques, rapports) et services dédiés (routes).

Le **chapitre 10** traitera exclusivement de l'authentification (flux, 2FA, emails), et le **chapitre 11** proposera un audit sécurité global (CSP, rate-limiting, surfaces d'attaque, secrets, etc.).

8.1. ARCHITECTURE GÉNÉRALE

Initialisation de l'application

Le fichier `backend/app.py` centralise la configuration (deux *binds* SQLAlchemy : `stops_db` et `auth_db`), les extensions (SQLAlchemy, CSRF, Limiter, Talisman, Migrate) et l'enregistrement des blueprints.

Décisions structurantes

- **Séparation des données** `stops_db` vs `auth_db` : isolement sécurité et privilèges distincts (Chap. 10), tout en simplifiant les migrations.
- **Extensions par défaut** activées dès l'`init` : CSRF (protection formulaire/AJAX), Limiter (défense anti-abus), Talisman (en-têtes de sécurité), Migrate (évolution du schéma sans downtime).
- **Organisation par domaines** via blueprints : `data`, `search`, `stats`, `reports`, `problems`, `auth`. Chaque route reste courte et testable.
- **Configuration 12-factor** par variables d'environnement (avec valeurs raisonnables en dev ; cf. Chap. 12 pour Compose).

Modèles et sérialisation

Les tables applicatives sont décrites dans `backend/models.py` (cf. Chap. 7 pour le schéma). La sérialisation cohérente d'un Stop en JSON est gérée par `backend/serializers/stops.py`.

Contrat JSON minimal d'un Stop

- **Identité** : `id`, `sloid`, `osm_node_id`
- **Type** : `stop_type` (`matched`|`unmatched`|`osm`), `match_type` (`auto`|`manual`)
- **Position** : `atlas_lat`/`lon` avec `fallback OSM` si manquant; `osm_lat`/`lon`

13. Flask. Documentation [en ligne]. Disponible sur : <https://flask.palletsprojects.com/> (consulté le 2025-08-05).

- **Métadonnées** : `distance_m`, `uic_ref`, `osm_node_type`
- **Invariants** : toujours renvoyer des coordonnées exploitables pour le rendu ; éviter les JOIN au point de clic (détails chargés à la demande)

8.2. BLUEPRINTS ET ENDPOINTS

Nous structurons l’API par *domaine* pour garder des fichiers courts, testables et lisibles.

a. Données : /api/data

Fichier : backend/blueprints/data.py. Endpoint principal pour alimenter la carte.

Rôle

- Filtrer par **fenêtre cartographique** (`bbox`), **types de transport**, **opérateurs**, **lignes/directions**
- Paginer et renvoyer un **JSON compact** prêt à dessiner
- **Rate limiting** : 30/min

Pourquoi c'est rapide

- Schéma de Chap. 7 : une ligne `stops` suffit pour un marqueur (pas de JOIN)
- Filtrage *SARGable* sur indexes lat/lon ATLAS et OSM
- Sérialisation minimale et pagination systématique

b. Recherche et matches manuels

Fichier : backend/blueprints/search.py. Recherches textuelles et *matches* manuels.

Endpoints clés

- `/api/search` : recherche par nom, opérateur, UIC
- `/api/top_matches` : tri par distance croissante selon filtres actifs
- `/api/manual_match` (POST) : crée un *match* ATLAS ↔ OSM, avec option de persistance

Persistance des matches

- Marque les deux entrées comme `matched` avec `match_type = manual`
- Si demandé, enregistre des marqueurs persistants pour survivre aux ré-imports (cf. Chap. 7)

c. Statistiques : /api/global_stats

Fichier : backend/blueprints/stats.py. Statistiques agrégées avec **cache LRU**.

Choix de conception

- Clé de cache *canonique* construite à partir des filtres triés
- Capacité bornée (50) avec éviction LRU et verrou léger pour la concurrence
- **Rate limiting** 30/min pour éviter le sur-calculation

d. Rapports : /api/generate_report

Fichier : backend/blueprints/reports.py. Génération PDF/CSV (pdfkit).

Principes

- **Auth requise et quotas** (20/jour) pour limiter l'abus
- Requêtes **optimisées par colonne** : seules les colonnes nécessaires sont sélectionnées
- Deux chemins : rendu template HTML → PDF ou export CSV avec en-têtes adaptés

e. Problèmes et persistance

Fichier : backend/blueprints/problems.py. Consultation, filtrage, tri, et persistance des solutions/notes. (Chap. 7 détaille la persistance côté import et modèle ; ici nous couvrons l'API.)

Capacités

- /api/problems : liste paginée (tri par priorité puis distance)
- /api/problems/stats : totaux résolus/non résolus par type
- /api/save_solution et /api/save_note/<note_type> : écriture simple
- /api/make_solution_persistent et /api/make_note_persistent/<note_type> : persistance entre ré-imports

8.3. REQUÊTAGE PARTAGÉ ET OPTIMISATION

QueryBuilder et FilterBuilder

Fichier : backend/query_builder.py. Centralise les patrons de filtres (type de transport, type de nœud, opérateurs ATLAS, routes) et applique des options de chargement si nécessaire.

Filtres pris en charge

- **Transport** : station, platform, stop_position, etc. (via sous-requêtes sur osm_nodes)
- **Opérateurs ATLAS** : filtre sur atlas_stops.operator
- **Routes/directions** : via routes_and_directions pour alimenter la UI

Invariants de composition

- Les conditions de transport sont agrégées par OR, puis combinées avec le reste par AND
- Paramètres homogènes et stables côté API : transport_types, operators, bbox, etc.

Services de routes

Fichier : backend/services/routes.py. Service get_stops_for_route(route_id, direction) en SQL brut sur routes_and_directions.

Points clés

- **Recherche tolérante** : correspondance par LIKE sur osm_route_id, atlas_route_id ou atlas_line_name
- **Fallback** de normalisation (ex. : - j24 → - jXX) ; récursif tant que l'ID se simplifie
- **Charge minimale** : sélection d'une ligne LIMIT 1 indexée, et désérialisation JSON côté Python

8.4. DIAGRAMMES DE FLUX

Flux d'une requête cartographique

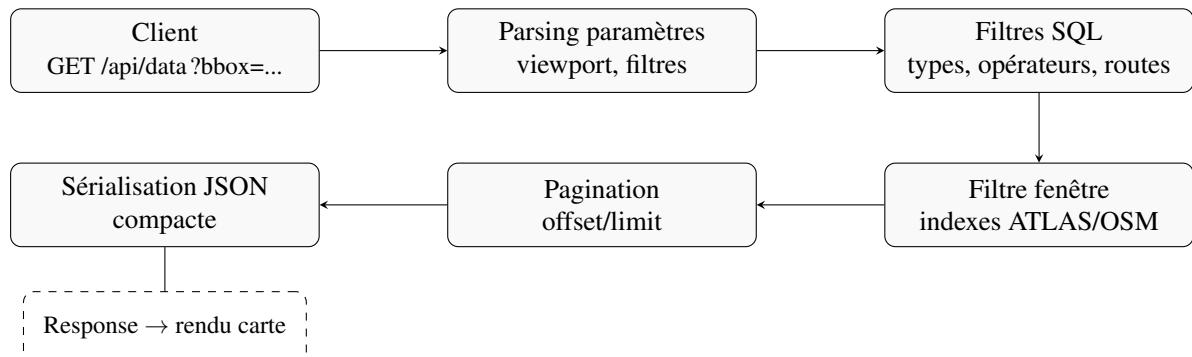


ILLUSTRATION 8.1 – Pipeline de traitement de /api/data : paramètres → filtres → fenêtre indexée → pagination → JSON.

Cycle de persistance des solutions

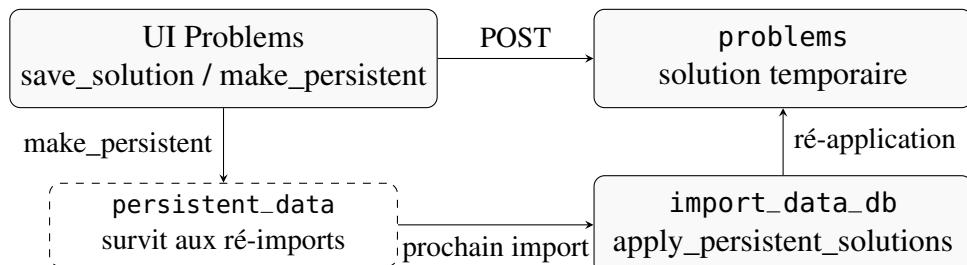


ILLUSTRATION 8.2 – Persistance des solutions/notes : écriture côté API, stockage durable, ré-application au prochain import.

8.5. BONNES PRATIQUES ET LISIBILITÉ

La maintenabilité du backend repose sur plusieurs principes de conception :

Endpoints courts et focalisés Chaque route a une responsabilité claire et unique. Les blueprints organisent les fonctionnalités par domaine métier.

Paramètres homogènes Noms stables et prédictibles à travers l'API (ex : stop_filter, match_method, bbox) pour simplifier l'intégration côté client.

Sérialisation centralisée Utilisation systématique de `format_stop_data()` pour éviter la divergence des formats JSON entre endpoints.

Rate limiting défensif Protection par défaut via `Limiter` avec réglages fins par route selon la criticité (ex : 30/min pour `/api/data`, 5/min pour les rapports).

Optimisations anti-patterns Éviter `ORDER BY RAND()` sur gros volumes : préférer une sélection pseudo-aléatoire par plage d'ID (cf. `/api/random_stop`).

Documentation Chaque endpoint inclut une docstring explicative pour faciliter la génération automatique de documentation API.

ET ENSUITE ?

Nous approfondirons les mécanismes d'authentification et d'autorisation au **chapitre 10**, puis l'ensemble du durcissement de la surface d'attaque au **chapitre 11**.

CHAPITRE 9 : FRONTEND

PANORAMA

Ce chapitre présente l'architecture du **frontend** et l'ergonomie de l'application. Il se lit comme un guide de visite : la carte interactive (page d'accueil), l'outil d'identification des problèmes, la gestion des données persistantes, et la génération de rapports. Pour les aspects API et modèles de données, voir **Chapitre 8**. Ici nous décrivons le cycle de vie des requêtes, les seuils de zoom, le rendu des marqueurs et la logique des info-bulles.

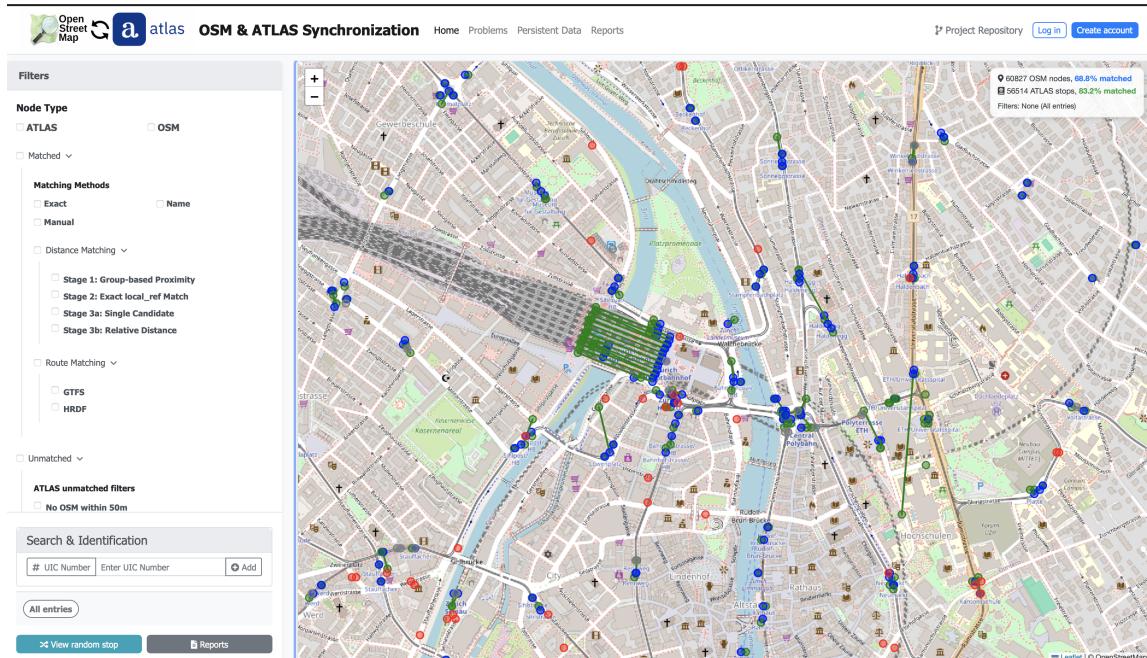


ILLUSTRATION 9.1 – Vue générale de la page d'accueil

9.1. PAGES ET NAVIGATION

L'application comporte quatre vues principales :

- **Carte (Index)** — exploration des arrêts, filtres riches, info-bulles, et un *Top N* des plus grandes distances. (Template : templates/pages/index.html)
- **Problèmes** — tri, filtrage et résolution guidée des anomalies avec contexte cartographique local. (templates/pages/problems.html)
- **Données persistantes** — revue et gestion des solutions/notes persistées entre imports. (templates/pages/persistent_data.html)
- **Rapports** — page dédiée de configuration et rendu PDF/CSV via backend (templates/pages/reports.html, cf. §9.5).

ILLUSTRATION 9.2 – Filtre ATLAS activé — ville de Zurich

Chaque page est construite en HTML Jinja¹⁴, stylisée par les feuilles CSS du dossier static/css, et animée par du JavaScript modulaire dans static/js.

9.2. CARTE INTERACTIVE (INDEX)

a. Cycle de vie des requêtes

La carte (Leaflet¹⁵) initialise une vue par défaut et **anti-rebondit** les événements moveend/zoomend (~ 320 ms) avant de charger la nouvelle fenêtre d'affichage. Les requêtes en vol sont **annulées** lorsque l'utilisateur se déplace à nouveau, ce qui évite les rendus obsolètes. Les paramètres envoyés à l'API incluent la *bbox* et les filtres actifs. Références côté serveur : /api/data and /api/stop_popup (voir Chap. 8 pour la logique et la sérialisation).

- **Seuils de zoom** — pas de marqueurs en dessous de $z < 13$; les *polylinnes* de liaison ne s'affichent qu'à partir de $z \geq 14$.
- **Petits ensembles à faible zoom** — une *sonde* plafonnée à ≤ 250 entrées permet d'afficher un petit ensemble même à bas zoom; sinon, une bannière invite à zoomer.

14. Jinja. Jinja Documentation [en ligne]. Disponible sur : <https://jinja.palletsprojects.com/> (consulté le 2025-08-06).

15. Leaflet. Leaflet Documentation [en ligne]. Disponible sur : <https://leafletjs.com/> (consulté le 2025-08-07).

- **Milieu de zoom** — résultats plafonnés (≤ 500).
- **Fort zoom** — plafond levé : *tous* les marqueurs de la fenêtre sont renvoyés.

Capture d'écran : bannière “zoomez” et apparitions des marqueurs.

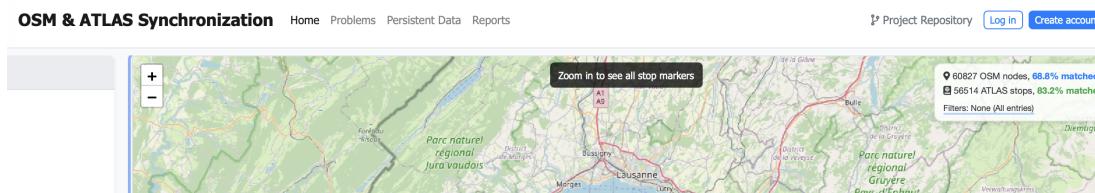


ILLUSTRATION 9.3 – Bannière politique de rendu selon le niveau de zoom

b. Rendu des marqueurs et des lignes

Le rendu privilégie des cercles Canvas/SVG légers jusqu’au zoom $z < 18$. Au-delà, certaines icônes *lettrees* (D, P, S) sont utilisées pour signaler *duplicate*, *platform*, *station*. Les icônes DOM identiques sont **mises en cache** pour éviter les reconstructions inutiles.

Lorsque des marqueurs se superposent exactement, un **décalage circulaire** subtil est appliqué pour éviter l’occlusion. Les ajouts au calque sont **lotis** ($\sim 150\text{--}200$ par lot) afin de préserver la réactivité du thread principal.

Les paires appariées (ATLAS–OSM) peuvent afficher une **ligne de liaison** (verte par défaut ; violette pour un match manuel, en pointillés si non persistant) lorsque les deux extrémités sont visibles et que le zoom le permet.

c. Info-bulles et chargement paresseux

Le contenu des popups est rendu par un **moteur de gabarits côté client** (PopupRenderer). Un premier clic déclenche un appel à /api/stop_popup qui renvoie des détails enrichis (noms, opérateurs, routes, notes). Les vues *unifiées* permettent d’inspecter **toutes les correspondances** d’un nœud (ATLAS vs OSM) sans recharger la page. Les popups sont déplaçables et conservent une *ligne d’ancrage* lors des déplacements/zooms.

Pour les entrées non appariées, un bouton **Match to** apparaît dans l’info-bulle. La sélection d’une cible opposée (ATLAS \leftrightarrow OSM) déclenche la création d’un match manuel via /api/manual_match. Cette action est cohérente avec le *workflow* décrit au Chap. 8 (persistance optionnelle).

d. Panneau de filtres et recherche

Le panneau de gauche agrège des filtres **par domaines fonctionnels** :

- **Type de nœud** (ATLAS/OSM) et **Type d’arrêt** (*Matched*, *Unmatched*, *Station*).

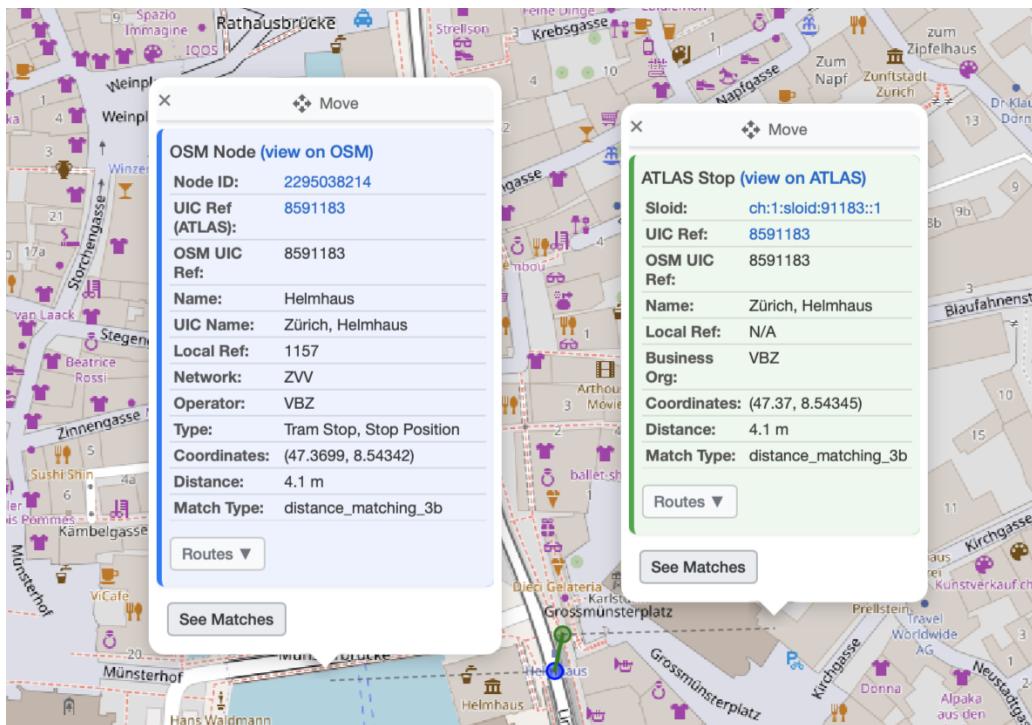


ILLUSTRATION 9.4 – Info-bulles

- **Méthodes d'appariement** : Exact, Name, Manual ; **Distance Matching** (stages 1, 2, 3a, 3b) ; **Route Matching** (GTFS, HRDF).
- **Top N Distances** — surcouche légère pour inspecter rapidement les plus grandes distances.
- **Transports** (station, platform, stop_position, ferry, aerialway, tram, etc.).
- **Opérateurs ATLAS** — dropdown dédié avec recherche (chargé depuis /api/operators).
- **Filtres spéciaux** — afficher uniquement les *duplicates* ATLAS.

La zone *Active Filters* affiche des "chips" cliquables (ET/OU) pour retirer rapidement un critère. Un **sélecteur de type de recherche** (numéro UIC, SLOID ATLAS, OSM Node ID, Route ID) ajuste dynamiquement le champ de saisie et les actions associées. Un **résumé d'en-tête** interroge /api/global_stats pour afficher des indicateurs utiles (ex. : pourcentage de nœuds appariés) synchronisés avec les filtres.

9.3. OUTIL PROBLÈMES

La page **Problèmes** combine : un panneau de filtres repliable (type d'anomalie, tri, opérateurs, priorité), une **carte de contexte** locale ($\pm 0.02^\circ$), et un **panneau de résolution** avec navigation par entrées. Les raccourcis clavier accélèrent la revue (\rightarrow /Espace : suivant, \leftarrow : précédent, \uparrow/\downarrow : défilement).

Les boutons d'action proposent des solutions pré-remplies (selon le type d'anomalie), la **persistance** des corrections, et l'édition de **notes** ATLAS/OSM. Le *toggle Auto-Persist* per-

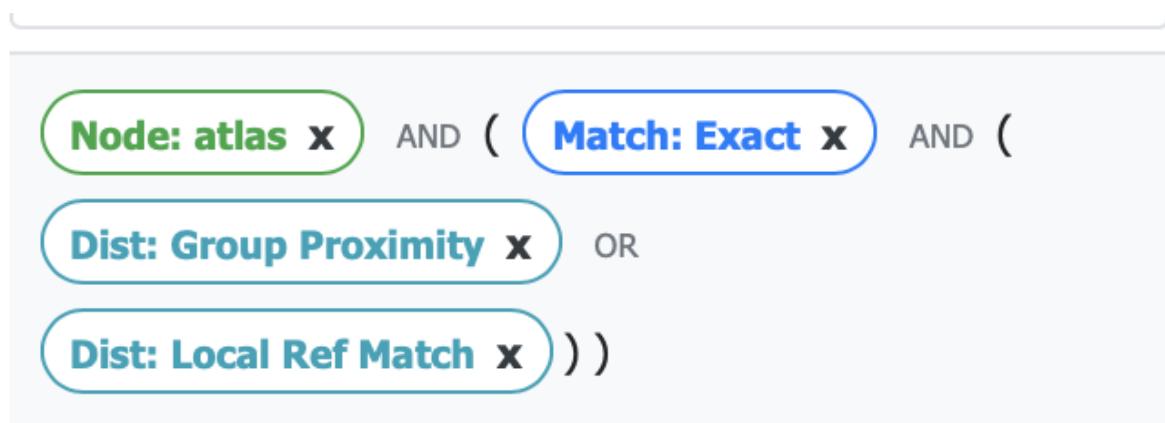


ILLUSTRATION 9.5 – Panneau de filtres avec « chips » actifs

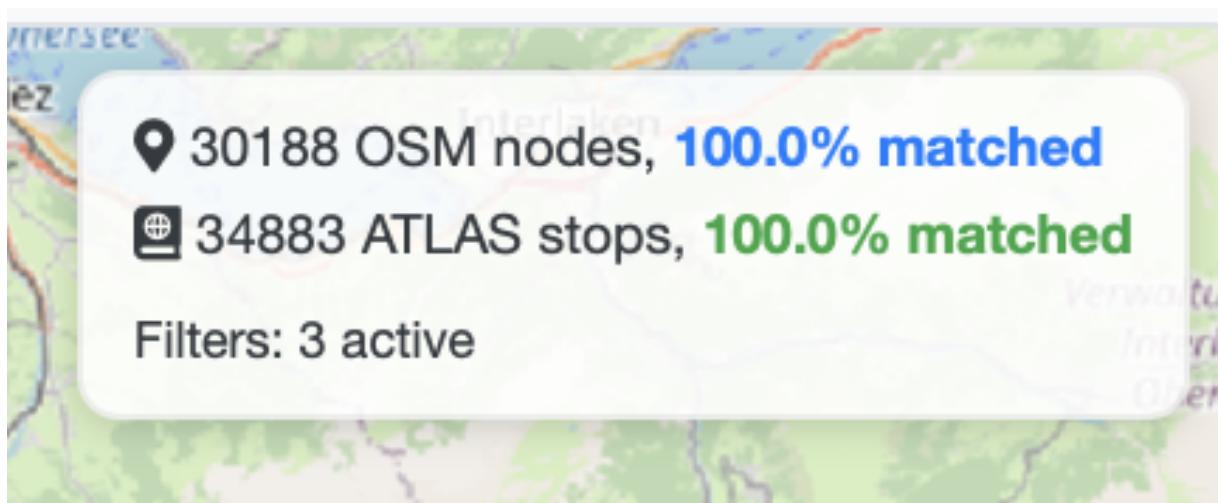


ILLUSTRATION 9.6 – Résumé d'en-tête avec statistiques globales. 3 filtres actifs

met de rendre persistantes toutes les nouvelles solutions/notes ; l'information est stockée pour être **réappliquée automatiquement** lors d'un prochain import (voir § *Persistance* et Chap. 8).

9.4. DONNÉES PERSISTANTES

La page **Données persistantes** distingue les *persistentes* (appliquées après import) des *non persistentes*. On peut filtrer par type (distance, unmatched, attributes, notes) et effectuer des actions massives (*Make All Persistent*, suppression/vidage). Côté serveur, ces opérations appellent les endpoints dédiés décrits au Chap. 8.

9.5. RAPPORTS ET INSTANTANÉS

Une **page dédiée** (/reports) propose un formulaire de configuration pour générer des rapports (*Top Matches*, *Exact*, *Name*, *Duplicates*) en PDF ou CSV via l'endpoint /api/generate_report (voir Chap. 8). Un **plafond de 20 téléchargements par IP et par jour** est appliqué côté serveur (limiteur global avec règle spécifique à l'endpoint) pour garan-

The screenshot displays the 'OSM & ATLAS Synchronization' application's 'Problems' page. On the left, there's a sidebar with 'Problem Filters' (Priority: P1, P2, P3) and 'Active Filters' (All Problems). Below that is the 'Atlas Operator' dropdown. The main area features a map of a road with a stop sign icon. A context overlay for 'ATLAS Stop (view on ATLAS)' provides details like StopID: ch1:stop:4:0:1374, UIC Ref: 8500004, OSM UIC Ref: N/A, Name: Selva, via Alpus spv., Local Ref: N/A, Business Org: MGB/Autoflo, Org: MGB/Autoflo, Coordinates: (46.6621, 8.72381), Distance: 8.2 m, Match Type: exact, and Routes. To the right, a detailed panel for 'Entry 1 of 52738 (1 Problem)' compares 'ATLAS Entry' and 'OSM Entry'. It shows that the operator differs between ATLAS and OSM. The 'ATLAS Stop' entry has StopID: ch1:stop:4:0:1374, UIC Ref: 8500004, Name: Selva, via Alpus spv., Local Ref: N/A, Business Org: MGB/Autoflo, Coordinates: (46.6621, 8.72381), Match Type: exact, and no route information available. The 'OSM Node' entry has Node ID: 7141379029, UIC Ref: 8500004, Name: Selva, via Alpus spv., UIC Name: Selva, via Alpus spv., Operator: MGB, Type: Platform, Coordinates: (46.6621, 8.72387), and Match Type: exact, also with no route information available. Below these are 'Resolution Actions' (choose correct source for mismatched attributes) and 'Overall Status' (Not a valid match, Skip). Persistence options at the bottom include Auto-Persist new solutions and Auto-Persist new notes.

ILLUSTRATION 9.7 – Page Problèmes — vue d’ensemble : filtres, carte de contexte et panneau de résolution

tir un usage équitable. Un **instantané de carte** (`map_snapshot.html`) peut afficher un groupe UIC/Opérateur avec marqueurs et liaisons, utile pour des annexes ou l’intégration dans le mémoire.

9.6. LIAISON AVEC LE BACKEND (RAPPEL CHAP. 8)

La cohérence et la réactivité de l’interface reposent sur :

- `/api/data` — charge utile **minimale** pour la navigation (coordonnées, types, distance, identifiants).
- `/api/stop_popup` — **détails** à la demande pour les info-bulles.
- `/api/top_matches`, `/api/global_stats` — **synthèses** parallèles non bloquantes.
- `/api/manual_match`, `/api/save_solution`, `/api/make_solution_persistent`, `/api/save_note` — actions **créatrices** assorties d’une option de persistance.

Ces endpoints, leurs limites de débit et la sérialisation `format_stop_data()` sont détaillés au Chapitre 8.

CONCLUSION

Le frontend associe une carte **fluide** (anti-rebond, annulation, seuils de zoom) à des **filtres expressifs** et à des info-bulles **paresseuses** pour conserver des charges utiles petites. L’outil Problèmes structure le travail de gestion des données et persistance des données rendant les corrections **durables** entre imports. La génération de rapports complète l’ensemble en offrant des exports soignés.

Problem Filters

Filter by Problem Type:

All Problems

Priority

- All
- P1 (selected)
- P2
- P3

	All Problems	17172
↳ ✓	All Solved	0
↳ !	All Unsolved	17172

Filtre de priorité et tri.

Atlas Operator:

Select operators...

Active Filters: Unmatched AND Priority P1

Persistence Options

Auto-Persist new solutions

Auto-Persist new notes

Manage Persistent Data

Chips actifs et options de persistance.

ILLUSTRATION 9.8 – Contrôles de filtrage et options de persistance — Page Problèmes

Entry 4 of 52738 (1 Problem)

Distance P3 (104m apart)

Resolution Actions

Distance between ATLAS and OSM: 104 m. Distance above 25 m for SBB. Choose which location is correct.

ATLAS correct OSM correct Both correct Not a match

Notes

ATLAS Note: Add a note for this ATLAS entry... OSM Note: Add a note for this OSM entry...

Buttons

Save ATLAS Note Save OSM Note Edit in OSM ID Editor

Actions proposées : problèmes de distance.

Entry 1 of 7538 (1 Problem)

Attribute Comparison

Critical attribute mismatch, UIC Name.

ATLAS Entry

ATLAS Stop (view on ATLAS)

StopID:	ch1:stopid:207:4:10
UIC Ref:	8500207
Name:	Solothurn
Local Ref:	10
Business Org:	SBB
Coordinates:	(47.2035, 7.54313)
Match Type:	exact
No route information available	

OSM Entry

OSM Node (view on OSM)

Node ID:	6551129268
UIC Ref (ATLAS):	8500207
OSM UIC Ref:	8500207
Name:	Solothurn RBS
UIC Name:	Solothurn [Gleis 9-10]
Local Ref:	10
Network:	Libero
Operator:	RBS
Type:	Stop Position
Coordinates:	(47.2035, 7.54337)
Match Type:	exact
No route information available	

Resolution Actions

For each mismatched attribute, choose the correct source.

Operator	SBB	RBS	Use ATLAS	Use OSM
UIC Name	Solothurn	Solothurn [Gleis 9-10]	Use ATLAS	Use OSM

Overall Status

Not a valid match Skip

Actions proposées : problèmes d'attributs.

ILLUSTRATION 9.9 – Panneau de résolution — actions contextuelles selon le type d'anomalie

The screenshot shows the 'Persistent Data' section of the application. At the top, there are navigation links: 'Home', 'Problems', 'Persistent Data', and 'Reports'. Below these are buttons for 'Back to Problems' and 'Map page'. On the right, there are buttons for 'Make All Persistent' and 'Filter by Type'. A green info box states: 'Persistent data is automatically applied after each data import. They help maintain permanent problem resolutions after data imports.' Below it, two tabs are shown: 'Persistent Data' (selected) and 'Non-Persistent Data'. A blue info box says: 'This data is permanently stored and will be automatically applied after future data imports.' A red error box at the bottom states: 'Error loading persistent data: UNAUTHORIZED'.

ILLUSTRATION 9.10 – Données persistantes — pour utilisateurs non administrateurs

The screenshot shows the 'Generate Report' form. At the top, there are links for 'Open Street Map', 'atlas', 'OSM & ATLAS Synchronization', 'Home', 'Problems', 'Persistent Data', and 'Reports'. On the right, there are links for 'Project Repository', 'Log in', and 'Create account'. The main form has a 'Generate Report' button and a 'Back to map' link. It includes a note: 'Downloads are limited to 20 reports per IP address per day to ensure fair use.' It has dropdown menus for 'Report Type' (set to 'Top Matches by Distance'), 'Report Format' (set to 'PDF'), and 'Number of Entries (N)' (set to '10'). It also has a 'Sort Order' dropdown set to 'Distance (Descending)' and a 'Generate' button.

ILLUSTRATION 9.11 – Page Rapports — génération (PDF/CSV) et limite quotidienne

CHAPITRE 10 : SYSTÈME D'AUTHENTIFICATION SÉCURISÉ

10.1. OBJECTIFS

Nous avons conçu et mis en œuvre un système d'authentification complet, aligné sur les bonnes pratiques actuelles.

10.2. APERÇU DE L'ARCHITECTURE

L'application utilise un schéma d'authentification dédié lié à une base de données MySQL distincte (`auth_db`). Comme on l'a vu, les données principales de l'application demeurent dans `stops_db`. Cette séparation réduit le rayon d'impact et garantit, entre autres, que la réimportation des données de transport public ne touche jamais aux identifiants des utilisateurs.

a. Composants

- **Stockage des mots de passe** : Argon2id (argon2-cffi)¹⁸, à forte consommation mémoire, salé, avec des paramètres spécifiques à chaque hachage.
- **Sessions** : Flask-Login avec cookies sécurisés (HttpOnly, SameSite=Lax ; indicateur Secure en production).
- **CSRF** : Protection CSRF de Flask-WTF pour tous les formulaires POST et les points de terminaison concernés.
- **Limitation de débit** : Les routes de connexion et d'inscription sont limitées (Flask-Limiter) afin d'atténuer la force brute.
- **Sécurité du transport** : Flask-Talisman fournit les en-têtes de sécurité ; application stricte de HTTPS.
- **2FA** : TOTP (compatible Google Authenticator)¹⁶ avec activation par QR code et codes de secours à usage unique. **Le secret TOTP est chiffré au repos** (Fernet)¹⁹ ; les codes de secours sont hachés (Argon2).
- **Verrouillage de compte** : Verrouillage progressif après des échecs répétés, avec temporation exponentielle.
- **Vérification d'email** : Liens signés (validité 48 h) envoyés à l'inscription et lors d'une connexion non vérifiée. La vérification est *optionnelle par défaut*.
- **CAPTCHA** : Cloudflare Turnstile¹⁷ sur /auth/register et /auth/login.

18. Argon2. Argon2 documentation [en ligne]. Disponible sur : <https://argon2-cffi.readthedocs.io/en/stable/> (consulté le 2025-06-17).

16. IETF. RFC 6238 : TOTP : Time-Based One-Time Password Algorithm [en ligne]. Disponible sur : <https://www.rfc-editor.org/rfc/rfc6238> (consulté le 2025-06-15).

19. Cryptography. Fernet (symmetric encryption) [en ligne]. Disponible sur : <https://cryptography.io/en/latest/fernet/> (consulté le 2025-06-18).

17. Cloudflare. Cloudflare Turnstile [en ligne]. Disponible sur : <https://www.cloudflare.com/products/>

- **Journalisation d'audit** : enregistrement des événements d'authentification dans auth_db.auth_events et émission simultanée de logs JSON sur stdout

10.3. MODÈLE DE DONNÉES (AUTH_DB)

La table users stocke les comptes utilisateurs avec les champs suivants :

TABLEAU 10.1 – Champs de la table users

Identité	email (unique)
Authentification	password_hash (Argon2id)
Rôles	is_admin
Vérification d'email	is_email_verified, email_verified_at, last_verification_sent_at
2FA	is_totp_enabled, totp_secret
Codes de secours	JSON de hachés Argon2
Hygiène de compte	created_at, updated_at, last_login_at
Verrouillage	failed_login_attempts, locked_until

Journalisation d'audit (auth_events)

Une table auth_events (même schéma auth_db) consigne les événements de sécurité :

TABLEAU 10.2 – Champs de la table auth_events

Type d'événement	event_type (<i>registration, login_success, login_failure, account_locked, 2fa_success, 2fa_failure, 2fa_enabled, 2fa_disabled, email_verified</i>)
Utilisateur	user_id (nullable), email_attempted (pour les échecs)
Contexte	ip_address, user_agent
Détails	metadata_json
Horodatage	occurred_at (UTC)

Modèle minimal (extrait)

```

1 class AuthEvent(db.Model):
2     __bind_key__ = 'auth'
3     __tablename__ = 'auth_events'
4     id = db.Column(db.Integer, primary_key=True)
5     user_id = db.Column(db.Integer, db.ForeignKey('users.id'), index=True)
6     email_attempted = db.Column(db.String(255), index=True)
7     event_type = db.Column(db.String(50), nullable=False, index=True)
8     ip_address = db.Column(db.String(45))
9     user_agent = db.Column(db.Text)
10    metadata_json = db.Column(db.Text)

```

turnstile/ (consulté le 2025-06-16).

```
11     occurred_at = db.Column(db.DateTime, default=utcnow, index=True)
```

Les mêmes événements sont émis au format JSON sur la sortie standard du service pour une collecte centralisée.

a. Parcours utilisateur et impact sur les données

Cette sous-section illustre, étape par étape, comment les actions de l'utilisateur se traduisent dans le schéma auth_db.

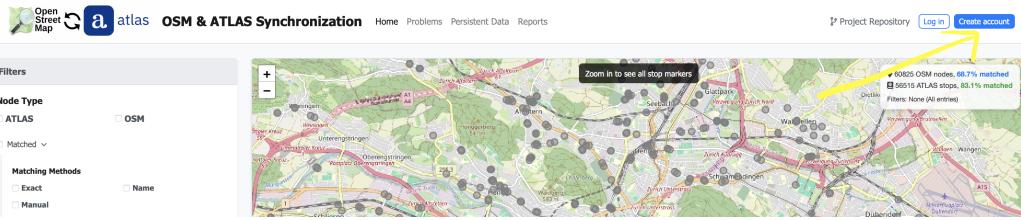


ILLUSTRATION 10.1 – Bouton « Créer un compte » visible dans l'interface.

Accès aux écrans d'authentification

Inscription Un utilisateur fournit un email et un mot de passe ; le serveur hache le mot de passe avec Argon2id et crée la ligne dans users.

Hachage du mot de passe (inscription)

```
1 from argon2 import PasswordHasher
2 ph = PasswordHasher()
3 password_hash = ph.hash(plain_password)
```

 A screenshot of the 'Create account' form. It has sections for 'What are accounts for?' (with options for saving problem solutions and requesting admin accounts), 'What are admin accounts for?' (with options for deleting persistent data), and a note about requesting admin privileges via email. Below are fields for 'Email' and 'Password', a checkbox for agreeing to the Terms of Service, and a 'Register' button. A small note at the bottom says 'We will email you a verification link (optional). You can verify later.' A yellow arrow points to the 'Create account' button at the top right of the page.

ILLUSTRATION 10.2 – Formulaire d'inscription (email, mot de passe).

	<code>id</code>	<code>email</code>	<code>password_hash</code>
1	1	bonjourguillem@gmail.com	\$argon2id\$v=19\$m=65536,t=3,p=4\$cdoqfUC84Y/6y60/nfRp...
2	2	bonjourguillem+1@gmail.com	\$argon2id\$v=19\$m=65536,t=3,p=4\$Yxp0+Ez+QqVdRztzRJg2...
3	3	bonjourguillem+2@gmail.com	\$argon2id\$v=19\$m=65536,t=3,p=4\$3vjxIrbnl9V8C9W7nPkL...

ILLUSTRATION 10.3 – auth_db.users : email et password_hash après inscription.

Vérification d'email À l'inscription, un lien de vérification signé (valide 48 h) est envoyé. Lorsqu'un utilisateur non vérifié se connecte avec succès, l'application **autorise la connexion** et renvoie un nouvel email de vérification accompagné d'un avertissement UI. Des routes dédiées existent pour *vérifier* (/auth/verify-email/<token>) et *renvoyer* (/auth/resend-verification) le lien. Cette vérification peut être rendue obligatoire côté produit si nécessaire.

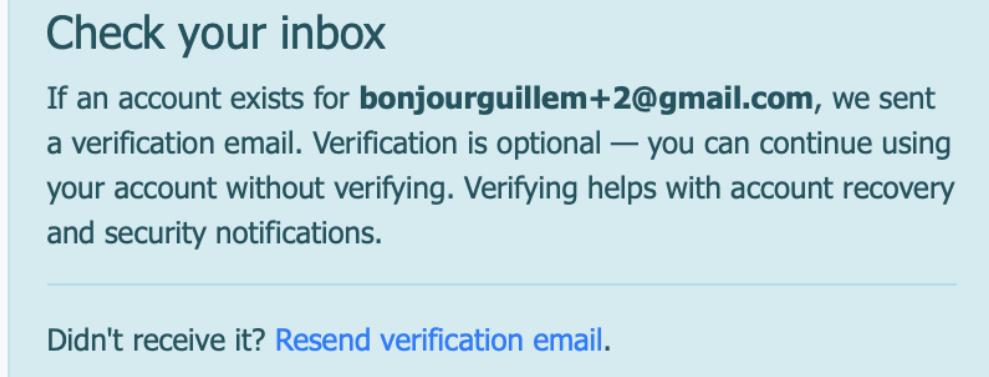


ILLUSTRATION 10.4 – Message après avoir demandé une vérification d'email.

Jeton de vérification d'email (extrait)

```
1 from itsdangerous import URLSafeTimedSerializer
2 s = URLSafeTimedSerializer(SECRET_KEY, salt="email-verification")
3 token = s.dumps({"uid": user_id})
4 # Plus tard: uid = int(s.loads(token, max_age=60*60*48)["uid"]) # 48 h
```

Délivrance des emails (Amazon SES)²⁰. Les emails de vérification sont rendus côté serveur puis envoyés via Amazon Simple Email Service (SES) grâce au SDK boto3. La fonction

20. Amazon Web Services. Amazon Simple Email Service (SES) [en ligne]. Disponible sur : <https://aws.amazon.com/ses/> (consulté le 2025-08-11).

`send_email(...)` centralise l'envoi dans `backend/services/email.py` et lit sa configuration via variables d'environnement :

- `AWS_REGION` (p. ex. `eu-west-1`)
- `SES_FROM_EMAIL` (identité SES vérifiée)
- `SES_CONFIGURATION_SET` (optionnel)

Le flux d'envoi est déclenché depuis `backend/blueprints/auth.py` : l'application génère le lien signé, rend les gabarits d'email (HTML et texte) dans `templates/emails/`, puis appelle `send_email(...)`. Les erreurs d'envoi sont journalisées mais n'interrompent pas le parcours utilisateur, afin d'éviter toute fuite d'information et de préserver l'ergonomie.

<code>is_admin</code>	<code>is_email_verified</code>	<code>email_verified_at</code>	<code>last_verification_sent_at</code>	<code>is_totp_enabled</code>
0	0 <null>		2025-08-16 10:31:04	1
0	0 <null>		2025-08-16 10:32:36	1
0	0 <null>		<null>	1

ILLUSTRATION 10.5 – `auth_db.users` : `is_admin`, `is_email_verified`, `last_verification_sent_at`, `is_totp_enabled`.

Connexion L'utilisateur s'authentifie sur la page de connexion ; en cas de succès, l'application met à jour l'historique de connexion et applique le verrouillage en cas d'échecs répétés.

ILLUSTRATION 10.6 – Page de connexion. Les limites de débit et le CAPTCHA (Turnstile) freinent les robots (voir Chap. 10).

ILLUSTRATION 10.7 – Connexion réussie : alertes UI sur l'email non vérifié et la 2FA non activée.

<input type="checkbox"/> last_login_at ↴	<input type="checkbox"/> failed_login_attempts ↴	<input type="checkbox"/> locked_until ↴
2025-08-16 10:31:04		0 <null>
2025-08-16 10:33:54		0 <null>
2025-08-16 18:02:12		0 <null>

ILLUSTRATION 10.8 – auth_db.users : last_login_at, failed_login_attempts, locked_until.

2FA Lorsqu'un utilisateur active la 2FA, le serveur génère un secret Base32 aléatoire et affiche un QR code contenant une URI *otpauth* standard. **Ce secret est chiffré au repos** (*cryptography.Fernet*) avec une clé d'application fournie par variable d'environnement, puis stocké dans auth_db.users.totp_secret. L'utilisateur vérifie le premier code à 6 chiffres pour activer la 2FA. Le serveur génère 10 codes de secours à usage unique et n'en stocke que les versions hachées avec Argon2. À la connexion, si la 2FA est active, l'utilisateur doit fournir un TOTP valide ou un code de secours non utilisé.

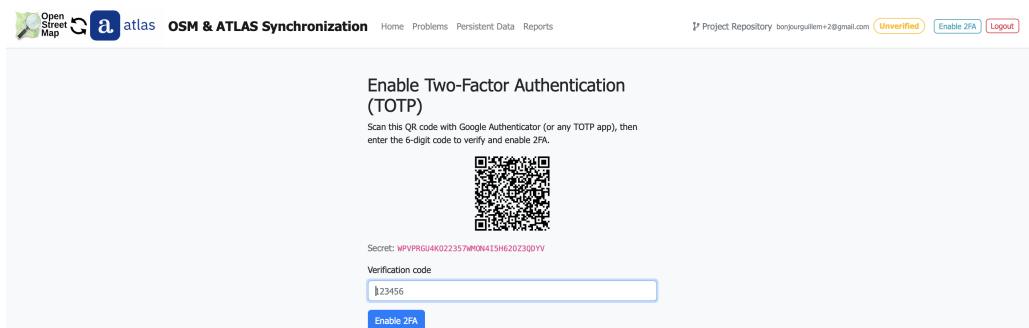


ILLUSTRATION 10.9 – Activation 2FA avec QR *otpauth* :// et secret Base32. QR et secret encodent la même information.

Validation TOTP (connexion avec 2FA)

```

1 import pyotp
2 secret = user.get_totp_secret() # déchiffre à la volée si chiffré
3 is_valid = pyotp.TOTP(secret).verify(code, valid_window=1)

```

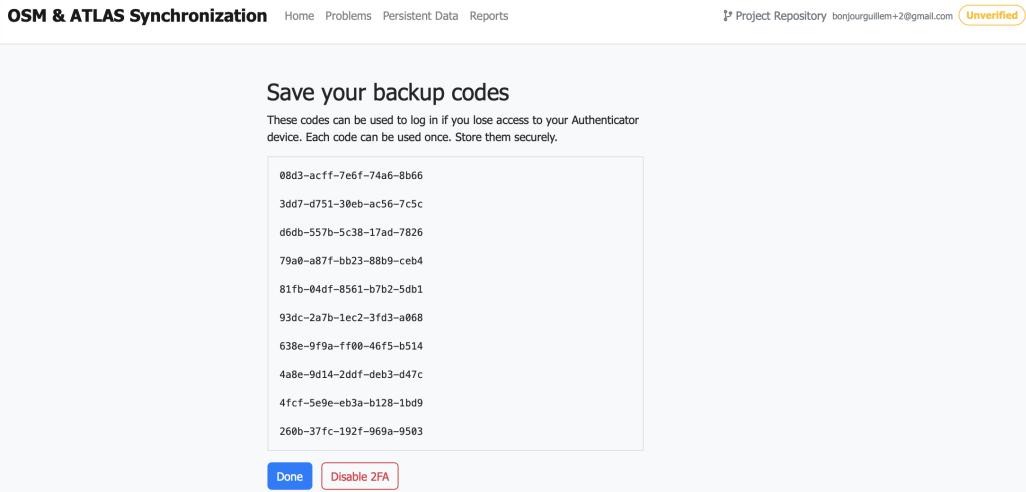


ILLUSTRATION 10.10 – Codes de secours : affichés une seule fois à l’activation. Stockage côté serveur : hachés Argon2 dans un JSON.

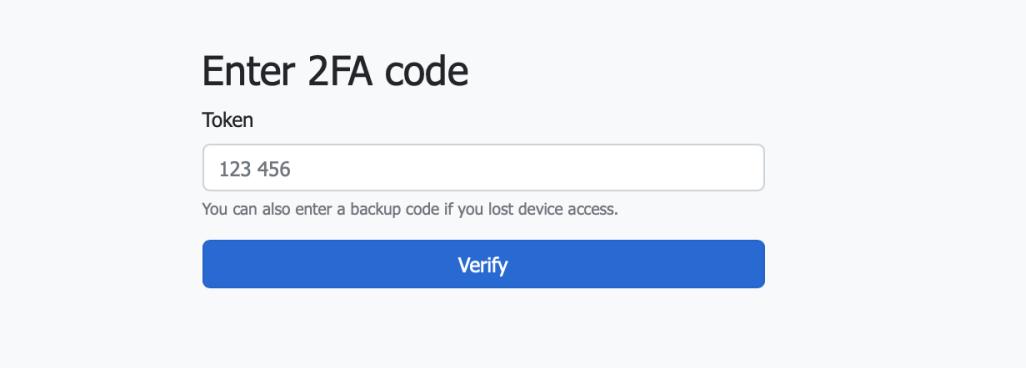


ILLUSTRATION 10.11 – Étape de connexion avec saisie du code 2FA (ou d’un code de secours).

<input type="checkbox"/> totp_secret	<input type="checkbox"/> backup_codes_json	<input type="checkbox"/> created_at	<input type="checkbox"/> updated_at
25CX4EDGL6IKNR4JHX6S0MD0R7BBNUAU	[{"argon2id": "v=19\$m=65536,t=3,p=4\$hMOHhJTXSD/m1bQCU...", "created_at": "2025-08-16 10:24:13", "updated_at": "2025-08-16 10:31:54"}]		
IKDNN23CH27TG3BGD5544K6KJFQH2JB	[{"argon2id": "v=19\$m=65536,t=3,p=4\$8Rasws14ZepM39KoXP...", "created_at": "2025-08-16 10:32:36", "updated_at": "2025-08-16 10:33:54"}]		
WPVPRGU4K022357WMON4I5H620Z3QDYY	[{"argon2id": "v=19\$m=65536,t=3,p=4\$0vg9N1S7LjEImYbcL...", "created_at": "2025-08-16 18:01:28", "updated_at": "2025-08-16 18:03:11"}]		

ILLUSTRATION 10.12 – auth_db.users : totp_secret (chiffré), backup_codes_json (haché), created_at, updated_at.

10.4. SÉCURITÉ OPÉRATIONNELLE (EN PRATIQUE)

Cette section résume les mesures concrètes d’exploitation sécurisée et l’intention derrière chacune.

- **Secrets et configuration :** Les clés d’application (SECRET_KEY), URL de base (AUTH_DATABASE_URI) et HTTPS sont fournis par variables d’environnement (Docker Compose). Ils ne sont jamais versionnés.

- **Mots de passe et codes** : Rien n'est stocké en clair. Les mots de passe sont hachés (Argon2id) ; les codes de secours sont également hachés.
- **Moindre privilège** : auth_db utilise des comptes dédiés avec des droits minimaux ; chaque service n'accède qu'au strict nécessaire.
- **Protection contre les attaques** : Limitation de débit, verrouillage progressif et CAPT-CHA atténuent la force brute et le bourrage d'identifiants.
- **En-têtes et CSP** : Talisman applique les en-têtes de sécurité et une politique CSP. Cette CSP peut être durcie en réduisant l'usage de CDN.
- **Traçabilité** : Les événements d'authentification sont consignés dans auth_events et envoyés en logs JSON pour la supervision.

CHAPITRE 11 : ÉVALUATION DE LA SÉCURITÉ DE L'APPLICATION

Ce chapitre complète le **Chap. 10** en évaluant les défenses mises en place, en illustrant les scénarios d'attaque plausibles et en discutant des améliorations futures. L'objectif est de comprendre comment et pourquoi les contrôles bloquent les attaques.

11.1. PÉRIMÈTRE ET SURFACE D'ATTAQUE

Pour rendre la lecture concrète, nous partons d'une *carte des routes* — celles que l'utilisateur peut réellement atteindre. Nous indiquons les garde-fous (authentification, limites de débit) directement à côté de chaque groupe.

Pages (UI) /, /map_snapshot, /problems, /persistent_data, /reports — vues HTML publiques qui appellent l'API ci-dessous.

Authentification (limitées par IP)

- /auth/register *GET,POST* — **5/min**; CAPTCHA Turnstile
- /auth/login *GET,POST* — **10/min**; CAPTCHA + verrouillage progressif par compte
- /auth/2fa *GET,POST* — **15/min**
- /auth/logout *POST* — **auth requis**
- /auth/enable_2fa *GET,POST* — **auth requis**
- /auth/disable_2fa *POST* — **auth requis**
- /auth/verify-email/<token> *GET* — **30/h**
- /auth/resend-verification *GET,POST* — **5/min** (réponse neutre)
- /auth/status *GET*

Données (lecture cartographique)

- /api/data *GET* — **30/min**; filtrage *SARGable* sur la fenêtre bbox, pagination
- /api/stop_popup *GET* — **120/min**; jointures ciblées, sérialisation compacte
- /api/route_stops *GET* — **60/min**; requête optimisée + fallback de normalisation
- /api/operators *GET* — **60/min**; SELECT DISTINCT indexé

Recherche

- /api/search *GET* — **60/min**; colonnes réduites via `optimize_query_for_endpoint`
- /api/top_matches *GET* — **60/min**; tri sur colonne indexée `distance_m`
- /api/random_stop *GET* — **30/min**; *sampling* par plages d'ID (pas de ORDER BY `RAND()`)

- /api/stop_by_id *GET* — **60/min**
- /api/manual_match *POST* — **30/min**; enregistre un appariement manuel

Statistiques et rapports

- /api/global_stats *GET* — **30/min**; cache LRU en mémoire
- /api/generate_report *GET* — **auth requis, 20/jour**; génération PDF/CSV côté serveur

Problèmes et persistance

- /api/problems, /api/problems/stats *GET* — **120/min**
- /api/save_solution, /api/make_solution_persistent *POST* — **30/min, auth requis**
- /api/save_note/atlas, /api/save_note/osm, /api/make_note_persistent/<type> *POST* — **60/min, auth requis**
- /api/check_persistent_solution, /api/check_persistent_note/... *GET* — **120/min**
- /api/persistent_data, /api/non_persistent_data *GET* — **60/min, auth requis**
- /api/persistent_data/<id> *DELETE* — **30/min, auth+admin**
- /api/make_non_persistent/<id>, /api/clear_all_persistent, /api/clear_all_non_persistent, /api/make_all_persistent *POST* — **10/h, auth requis** (admin si destructif)

Important. Les opérations *destructives* sur les données persistantes (comme *DELETE* /api/persistent_data/<id>) sont réservées aux **administrateurs**. Un utilisateur non admin ne peut ni supprimer ni vider ces jeux de données.

11.2. CONTRÔLES EN PLACE

Mots de passe et stockage

Les mots de passe sont hachés avec Argon2id¹⁸ via argon2-cffi, avec paramètres adaptés à la mémoire. Vérification sûre :

Vérifier un mot de passe

```
1 from argon2 import PasswordHasher
2 ph = PasswordHasher()
3 ph.verify(user.password_hash, candidate)
```

18. Argon2. Argon2 documentation [en ligne]. Disponible sur : <https://argon2-cffi.readthedocs.io/en/stable/> (consulté le 2025-06-17).

Sessions et cookies

Cookies `HttpOnly`, `SameSite=Lax` et `Secure` activable par variable d'environnement (en production : à forcer). Durée « remember » de 14 jours.

Paramètres de session (extrait)

```
1 app.config['SESSION_COOKIE_HTTPONLY'] = True
2 app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
3 app.config['SESSION_COOKIE_SECURE'] = os.getenv('SESSION_COOKIE_SECURE', 'false').
lower()=='true'
```

CSRF

Qu'est-ce que le CSRF? Une page malveillante pourrait faire envoyer à votre navigateur une requête POST à notre site en utilisant votre cookie de session, sans votre intention.

Comment on s'en protège. Les formulaires sensibles (inscription, connexion, 2FA, etc.) incluent un **jeton CSRF** caché généré par Flask-WTF. À chaque soumission, le serveur vérifie que le jeton du formulaire correspond à celui stocké côté session. Sans ce jeton, la requête est refusée.

Qu'est exempté. Les endpoints **lecture seule** JSON (GET) restent exemptés. Tout endpoint qui **modifie** de l'état via cookies de session doit exiger un jeton CSRF.

`SameSite=Lax` aide mais ne suffit pas : le jeton CSRF est la garantie principale contre l'exécution non intentionnelle de requêtes authentifiées.

CAPTCHA Turnstile

Un CAPTCHA Cloudflare Turnstile¹⁷ protège l'inscription et la connexion.

Vérifier le CAPTCHA (simplifié)

```
1 secret = os.getenv('TURNSTILE_SECRET_KEY', '')
2 if not secret: return True # dev
3 resp = requests.post('https://.../siteverify', data={...})
4 ok = bool(resp.json().get('success'))
```

17. Cloudflare. Cloudflare Turnstile [en ligne]. Disponible sur : <https://www.cloudflare.com/products/turnstile/> (consulté le 2025-06-16).

Durcissement du conteneur

Principe du moindre privilège. Le service app s'exécute désormais en **utilisateur non-root** (USER app) avec des **permissions minimales** dans /app. Seuls /app/data et /app/.cache sont inscriptibles ; le reste du code est en lecture seule. En cas d'exécution de code arbitraire (RCE), l'impact est réduit par l'absence de droits root.

Volumes et UID/GID. Pour éviter des problèmes d'écriture sur le volume monté .:/app, l'UID/GID du compte app peut être aligné sur l'hôte via des *build args APP_UID/APP_GID* (voir Chap. 12).

Limitation de débit et verrouillage

Idée générale. Nous combinons deux couches :

- **Limites par route** (Flask-Limiter) — plafonds « X par minute/heure/jour » appliqués par défaut **par IP**.
- **Verrouillage par compte** — après plusieurs mots de passe erronés, le compte est temporairement bloqué (délai croissant) pour ralentir les attaques.

Comment les limites fonctionnent. Chaque route a un *quota* (ex. : 10/min). À dépassement, le serveur répond 429 Too Many Requests. Le plafonnement est appliqué **par IP**. En complément, un **verrouillage par compte** après plusieurs échecs empêche de contourner la seule limite IP.

Caveat proxies. Si l'application est derrière un proxy (NGINX/Cloudflare), il faut **configurer les proxys de confiance** pour lire correctement X-Forwarded-For. Sans cela, la limitation « par IP » peut devenir inefficace ou injuste, car toutes les requêtes semblent provenir d'une même IP.

Limites de débit par route

```
1 @limiter.limit("5/minute")    # /auth/register
2 @limiter.limit("10/minute")   # /auth/login
3 @limiter.limit("15/minute")   # /auth/2fa
4 @limiter.limit("30/hour")     # /auth/verify-email/<token>
5 @limiter.limit("30/minute")   # /api/data
6 @limiter.limit("120/minute")  # /api/stop_popup
7 @limiter.limit("60/minute")   # /api/search, /api/top_matches, /api/route_stops, /
                               api/operators
8 @limiter.limit("30/minute")   # /api/random_stop, /api/manual_match
9 @limiter.limit("30/minute")   # /api/global_stats (avec cache LRU)
10 @login_required; @limiter.limit("20/day")  # /api/generate_report
```

Verrouillage exponentiel (extrait)

```
1 if not user.verify_password(password):
2     user.failed_login_attempts += 1
3     if user.failed_login_attempts >= 5:
4         lock_minutes = min(60, 2 * user.failed_login_attempts + 5)
5         user.locked_until = utcnow() + timedelta(minutes=lock_minutes)
```

2FA TOTP et codes de secours

TOTP standard (30 s)¹⁶ via pyotp. **Le secret TOTP est chiffré au repos** (Fernet)¹⁹ et déchiffré à la volée à l'usage ; les codes de secours sont **hachés** (Argon2) et **consommés** à l'usage.

Vérifier TOTP ou code de secours

```
1 secret = user.get_totp_secret() # déchiffre si nécessaire
2 totp_ok = pyotp.TOTP(secret).verify(token, valid_window=1)
3 backup_ok = user.verify_and_consume_backup_code(token)
```

Journalisation d'audit

Chaque tentative ou succès d'action sensible est consignée dans auth_db.auth_events et émise en JSON dans les logs :

- **Événements** : inscription, connexion (succès/échec), verrouillage, 2FA (succès/échec), activation/désactivation 2FA, email vérifié, logout.
- **Contexte** : IP (X-Forwarded-For prioritaire), User-Agent, email tenté (si échec), métadonnées.
- **Exploitation** : requêtes SQL côté auth_db ou filtrage des logs docker compose logs.

Exemples de requêtes

```
1 -- Connexions échouées pour un email sur 24 h
2 SELECT occurred_at, ip_address, metadata_json
3 FROM auth_events
4 WHERE event_type = 'login_failure'
5 AND email_attempted = 'user@example.com'
6 AND occurred_at > NOW() - INTERVAL 1 DAY
7 ORDER BY occurred_at DESC;
```

16. IETF. RFC 6238 : TOTP : Time-Based One-Time Password Algorithm [en ligne]. Disponible sur : <https://www.rfc-editor.org/rfc/rfc6238> (consulté le 2025-06-15).

19. Cryptography. Fernet (symmetric encryption) [en ligne]. Disponible sur : <https://cryptography.io/en/latest/fernet/> (consulté le 2025-06-18).

Filtrer les événements dans les logs du conteneur

```
! docker compose logs -f app | grep 'auth_event' | cat
```

11.3. SCÉNARIOS D’ATTAQUE ET DÉROULÉ

A1 — Bourrage d’identifiants / force brute

Attaque. Un robot tente des listes « email+mot de passe ».

Côté serveur. La route `/auth/login` est plafonnée à 10/min par IP; les échecs incrémentent un compteur par compte. Après 5 échecs : verrouillage progressif (5, 7, 9, ... minutes jusqu’à 60 min max). Cookies non délivrés tant que la session n’est pas authentifiée.

Résultat. Les rafales sont ralenties par IP (limiteur) et par compte (verrouillage), ce qui rend l’attaque coûteuse et lente. Des détails supplémentaires de calibrage sont discutés ci-dessous.

A2 — Contournement 2FA

Attaque. L’attaquant dérobe un mot de passe (hameçonnage) et tente de se connecter sans 2FA.

Côté serveur. Si `is_totp_enabled=true`, une étape 2FA obligatoire s’intercale. Un TOTP valide (± 30 s) ou un code de secours non consommé est requis.

Résultat. Sans le second facteur (ou un code de secours), l’intrusion échoue. Les codes de secours étant **hachés** et **consommés**, leur ré-utilisation est impossible.

Note . Le secret TOTP est **chiffré au repos** dans `auth_db` (Fernet), puis déchiffré à la volée pour la vérification. Les codes de secours restent hachés (Argon2) et sont consommés à l’usage.

A3 — Énumération d’email

Attaque. Tester si un email existe via les messages d’erreur.

Côté serveur. `/auth/login` répond « Invalid credentials » dans tous les cas; `/auth/resend-verification` répond identiquement qu’un compte existe ou non. `/auth/register` adopte désormais une **réponse neutre** : que l'email existe ou non, l'utilisateur est redirigé vers l'avis de vérification; si un compte non vérifié existe, un email de vérification est renvoyé en arrière-plan. Aucune information n'est divulguée sur l'existence du compte.

Résultat. L'énumération est évitée sur login/resend, mais possible sur register. Voir § ?? pour uniformiser les messages.

A4 — Abus de liens signés (vérification d'email)

Attaques visées.

- **Rejeu** d'un vieux lien après vérification ou expiration.
- **Bruteforce** du jeton signé.
- **Inondation** d'envois de mails de vérification.

Mesures côté serveur.

- Jetons `itsdangerous` avec **sel dédiée** et **expiration 48 h**; taille et entropie suffisantes pour rendre le bruteforce irréaliste.
- Vérification idempotente : si l'email est déjà vérifié, le jeton est ignoré proprement.
- Plafonds : `/auth/verify-email/...` à **30/h** et `/auth/resend-verification` à **5/min**; côté application, **1 email/min par compte**.
- Journalisation d'audit des vérifications et des renvois pour repérer des abus.

Effet. Les relectures expirent rapidement ; les tentatives massives et le spam sont freinés par les limites.

A5 — CSRF sur routes sensibles

Attaque. Une page externe soumet en cachette un POST authentifié (ex. : désactiver 2FA, créer un objet) en s'appuyant sur le cookie de session de la victime.

Côté serveur. Les formulaires sensibles exigent un **jeton CSRF** valide (Flask-WTF). Les routes JSON **mutatrices** doivent elles aussi vérifier un jeton ou un en-tête spécifique côté client. Les routes de **lecture** (GET) restent exemptées.

Résultat. Sans jeton légitime, la requête est refusée même si le cookie de session est présent. SameSite=Lax réduit le risque mais ne remplace pas le jeton.

A6 — DoS sur endpoints lourds

Attaque. Bombarder `/api/data`, `/api/stop_popup` ou `/api/generate_report` pour saturer la base/le CPU.

Côté serveur. Nous avons ajouté ou resserré des limites : `/api/data 30/min`, `/api/stop-popup 120/min`, `/api/search/top_matches 60/min`, `/api/global_stats 30/min` (avec **cache LRU**), `/api/generate_report` désormais **authentifiée** et plafonnée à **20/jour**.

Requêtes SARGable et pagination. « SARGable » signifie que les filtres portent directement sur des **colonnes indexées** (par exemple `WHERE lon BETWEEN ... AND ... AND lat BETWEEN ... AND ...`) plutôt que sur des expressions non indexables. La base peut ainsi utiliser ses index et éviter des scans complets. La **pagination** (`LIMIT/OFFSET` ou *keyset pagination*) impose un **maximum** d'éléments traités par requête, ce qui crée des **bornes fortes** sur le temps CPU et les lectures disque.

Résultat. Un attaquant doit multiplier les IPs et comptes authentifiés pour maintenir une charge significative ; l'impact reste contenu. Les logs aident à repérer des rafales anormales.

11.4. CALIBRAGE ET LIMITES ACTUELLES

Les limites en place forment un socle solide mais peuvent être durcies :

- **Limiteur par IP** : efficace mais contournable via réseaux distribués ; on pourrait ajouter une clé composite (IP + email cible) et des « seaux » par utilisateur.
- **Verrouillage par compte** : robuste, mais il faut faire attention au déni de service ciblé (un adversaire peut « verrouiller » le compte d'une victime). Des *captcha* et *cooldowns* aident à mitiger.

11.5. CONCLUSION

L'architecture d'authentification posée au **Chap. 10** est saine : hachage robuste, 2FA réelle, captcha, limites et verrouillage. Ce chapitre a montré *comment* ces mécanismes résistent aux attaques usuelles et a mis en lumière des durcissements concrets pour atteindre un niveau « production ».

CHAPITRE 12 : DÉPLOIEMENT ET CONTENEURISATION

Ce chapitre présente la conteneurisation du projet, conçue pour un déploiement automatisé. Une seule commande initialise la base de données, exécute les scripts d'importation et lance l'application web sur `http://localhost:5001`.

Nous détaillerons le rôle des fichiers `dockerfile`, `docker-compose.yml`, `entrypoint.sh` et `.dockerignore`.

12.1. INTRODUCTION À DOCKER

Docker est une plateforme open-source qui automatise le déploiement, la mise à l'échelle et la gestion des applications en utilisant la conteneurisation²¹. Un conteneur est une unité logicielle légère et autonome qui inclut tout ce dont une application a besoin pour fonctionner : le code, les dépendances, les bibliothèques et les outils système.

Contrairement aux machines virtuelles qui virtualisent un système d'exploitation complet, les conteneurs partagent le noyau de l'hôte tout en maintenant l'isolation des processus. Cette approche offre plusieurs avantages : démarrage rapide (quelques secondes), empreinte mémoire réduite et portabilité accrue. Les conteneurs isolent les applications les unes des autres et de leur environnement, garantissant ainsi qu'elles fonctionnent de manière cohérente sur n'importe quelle infrastructure supportant Docker.

Pourquoi dockeriser ?

Trois raisons concrètes :

- **Reproductibilité** : même version de Python, mêmes dépendances système (`wkhtmltopdf`²², client MySQL), même environnement d'exécution.
- **Parité dev/prod** : une pile unique (app + base) réduisant les « ça marche sur ma machine ».
- **Bootstrap instantané** : une commande pour installer, migrer, télécharger/traiter les données et lancer le serveur.

12.2. LES QUATRE PIÈCES DU PUZZLE

dockerfile Image de l'application : base Python 3.9 `slim-bookworm`, dépendances `apt`, `pip`, copie du code et `ENTRYPOINT`.

docker-compose.yml Orchestration multi-services : db (MySQL) + app et app-dev (profils), volumes, variables d'environnement²³, `depends_on` avec `healthcheck`.

21. Docker. Documentation [en ligne]. Disponible sur : <https://docs.docker.com/> (consulté le 2025-08-12).

22. wkhtmltopdf. wkhtmltopdf [en ligne]. Disponible sur : <https://wkhtmltopdf.org/> (consulté le 2025-08-13).

23. Docker. Environment variables with Compose [en ligne]. Disponible sur : <https://docs.docker.com/compose/environment-variables/envvars/> (consulté le 2025-08-14).

entrypoint.sh Orchestrateur de démarrage côté app : attend MySQL, applique migrations, exécute scripts de données (selon flags), lance Flask.

.dockerignore Réduit le contexte de build (ignore les dossiers volumineux et la documentation), accélère et assainit l'image.

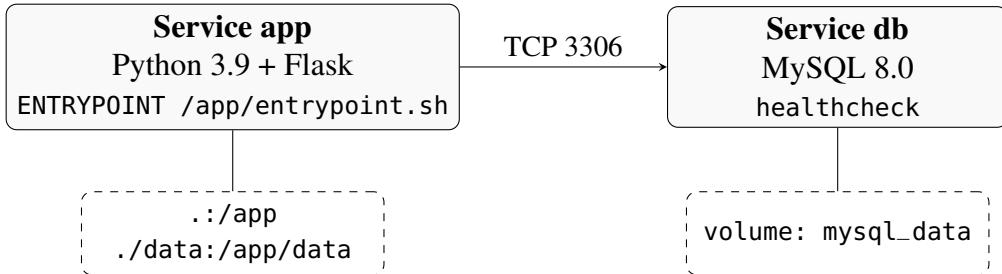


ILLUSTRATION 12.1 – Deux conteneurs reliés sur le réseau compose ; volumes pour le code (monté) et les données MySQL (persistées).

12.3. L'ORCHESTRATION : docker-compose.yml

Le fichier compose²⁴ définit db, app et un service app-dev activé via profil. Extraits commentés :

db : MySQL 8.0 avec healthcheck

```

1 services:
2   db:
3     image: mysql:8.0
4     environment:
5       MYSQL_ROOT_PASSWORD: root
6       MYSQL_DATABASE: stops_db
7       MYSQL_USER: stops_user
8       MYSQL_PASSWORD: 1234
9     ports:
10      - "3306:3306"
11     volumes:
12       - mysql_data:/var/lib/mysql
13     healthcheck:
14       test: ["CMD-SHELL", "mysqladmin ping -h localhost -u${MYSQL_USER} -p${MYSQL_PASSWORD}"]
15       interval: 10s
16       timeout: 5s
17       retries: 2

```

24. Docker. Docker Compose [en ligne]. Disponible sur : <https://docs.docker.com/compose/> (consulté le 2025-08-15).

L'application dépend explicitement du statut *healthy* de MySQL²⁵ et monte deux volumes (code + données) :

app : service principal

```
1 app:
2   build: .
3   ports:
4     - "5001:5001"
5   volumes:
6     - .:/app
7     - ./data:/app/data
8   depends_on:
9     db:
10       condition: service_healthy
11   environment:
12     # Variables lues depuis .env si présentes (avec valeurs par défaut)
13     MYSQL_USER: ${MYSQL_USER:-stops_user}
14     MYSQL_PASSWORD: ${MYSQL_PASSWORD:-1234}
15     AUTH_DB_USER: ${AUTH_DB_USER:-}
16     AUTH_DB_PASSWORD: ${AUTH_DB_PASSWORD:-}
17     DATABASE_URI: ${DATABASE_URI:-mysql+pymysql://stops_user:1234@db/stops_db}
18     AUTH_DATABASE_URI: ${AUTH_DATABASE_URI:-mysql+pymysql://stops_user:1234@db/
auth_db}
19     AUTO_MIGRATE: "true"
20     MATCH_ONLY: "${MATCH_ONLY:-false}"
21     SECRET_KEY: ${SECRET_KEY:-dev-insecure}
```

Pour développer l'application web, nous activons un service plus léger qui évite les téléchargements et prétraitements longs :

app-dev : profil dev

```
1 app-dev:
2   profiles: [dev]
3   build: .
4   volumes:
5     - .:/app
6     - ./data:/app/data
7   environment:
8     SKIP_DATA_IMPORT: "true"
9     AUTO_MIGRATE: "true"
```

²⁵. MySQL. MySQL 8.0 Documentation [en ligne]. Disponible sur : <https://dev.mysql.com/doc/> (consulté le 2025-08-16).

12.4. L'IMAGE : dockerfile

L'image part d'un Python 3.9 minimal `slim-bookworm` (compatibilité `wkhtmltopdf`). Nous installons plusieurs dépendances système critiques : `wkhtmltopdf` pour la génération PDF, `default-mysql-client` pour les commandes d'administration MySQL, `dos2unix` pour normaliser les fins de lignes, et les bibliothèques GDAL/GEOS nécessaires à GeoPandas pour le traitement géospatial. Puis `pip install -r requirements.txt` et utilisation d'un `ENTRYPOINT shell`.

Extraits — `dockerfile` (utilisateur non-root)

```

1 FROM python:3.9-slim-bookworm
2 ENV FLASK_APP=backend/app.py \\
3     FLASK_RUN_HOST=0.0.0.0 \\
4     FLASK_RUN_PORT=5001
5 RUN apt-get update && apt-get install -y --no-install-recommends \\
6     wkhtmltopdf default-mysql-client dos2unix && rm -rf /var/lib/apt/lists/*
7
8 # Crée un utilisateur non-root configurable
9 ARG APP_UID=1000
10 ARG APP_GID=1000
11 RUN groupadd -g ${APP_GID} app && \\
12     useradd -m -u ${APP_UID} -g ${APP_GID} -s /bin/bash app
13
14 WORKDIR /app
15 COPY requirements.txt /app/
16 RUN pip install --no-cache-dir -r requirements.txt
17 COPY entrypoint.sh /app/entrypoint.sh
18 RUN dos2unix /app/entrypoint.sh && chmod +x /app/entrypoint.sh
19 COPY . /app/
20
21 # Permissions minimales et répertoires d'écriture
22 RUN find /app -type d -exec chmod 755 {} \; && \\
23     find /app -type f -exec chmod 644 {} \; && \\
24     chmod 755 /app/entrypoint.sh && \\
25     mkdir -p /app/data /app/.cache && \\
26     chown -R app:app /app && \\
27     chmod 775 /app/data /app/.cache
28
29 # Exécute en tant qu'utilisateur non-root
30 USER app
31
32 EXPOSE 5001
33 ENTRYPOINT ["/bin/bash", "/app/entrypoint.sh"]

```

12.5. LE CHEF D'ORCHESTRE : `entrypoint.sh`

Le script d'entrée synchronise le démarrage : attend MySQL, crée la base auth_db si besoin, applique les migrations, exécute les scripts de données (selon les drapeaux), puis lance Flask¹³. Il peut également créer un utilisateur dédié pour auth_db si des variables d'environnement sont fournies.

Attente active, création de comptes, migrations

```

1 echo "Waiting for MySQL database at db:3306..."
2 while ! mysqladmin ping -h"db" -P3306 --silent \\
3         --user=${MYSQL_USER} --password=${MYSQL_PASSWORD}; do
4     sleep 1
5 done
6 echo "MySQL is up and ready."
7
8 if [ -n "$MYSQL_ROOT_PASSWORD" ]; then
9     # Assure l'existence de auth_db et droits de base
10    mysql -h db -uroot -p"${MYSQL_ROOT_PASSWORD}" -e "
11        CREATE DATABASE IF NOT EXISTS auth_db
12            CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
13        GRANT ALL PRIVILEGES ON auth_db.* TO 'stops_user'@'%';
14        FLUSH PRIVILEGES;" || true
15
16 # Crée un utilisateur dédié auth si variables fournies
17 if [ -n "$AUTH_DB_USER" ] && [ -n "$AUTH_DB_PASSWORD" ]; then
18    mysql -h db -uroot -p"${MYSQL_ROOT_PASSWORD}" -e "
19        CREATE USER IF NOT EXISTS '${AUTH_DB_USER}'@'%'
20            IDENTIFIED BY '${AUTH_DB_PASSWORD}';
21        GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, INDEX
22            ON auth_db.* TO '${AUTH_DB_USER}'@'%';
23        REVOKE ALL PRIVILEGES ON auth_db.* FROM 'stops_user'@'%';
24        FLUSH PRIVILEGES;" || true
25 fi
26 fi
27
28 if [ "${AUTO_MIGRATE:-false}" = "true" ]; then
29     [ ! -d "migrations" ] && flask db init || true
30     flask db migrate -m "Auto migration" || true
31 fi
32 flask db upgrade || true

```

Le démarrage exécute également `create_auth_tables.py` pour s'assurer que le schéma

13. Flask. Documentation [en ligne]. Disponible sur : <https://flask.palletsprojects.com/> (consulté le 2025-08-05).

auth_db inclut **users** et **auth_events** (journalisation d'audit).

12.6. DÉMARRER EN 30 SECONDES

Commande

```
1 docker compose up -d
```

Attendez quelques secondes ; l'app est disponible sur <http://localhost:5001>. Pour le mode développement (sans préparation de données) :

Commande

```
1 docker compose --profile dev up -d
```

Pour consulter les logs et vérifier que tout s'enchaîne bien :

Commande

```
1 docker compose logs -f app | cat
```

Pour ne suivre que les *événements d'authentification* structurés (JSON), filtrez :

Commande

```
1 docker compose logs -f app | grep auth_event | cat
```

```
[(base) MacBookPro:bachelor-project guillemmassague$ docker compose up
[+] Running 2/2
  ✓ Container bachelor_project_db   Created
  ✓ Container bachelor_project_app Recreated
Attaching to bachelor_project_app, bachelor_project_db
bachelor_project_db  | 2025-08-17 08:42:48+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.43-1.el9 started.
bachelor_project_db  | 2025-08-17 08:42:49+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
bachelor_project_db  | 2025-08-17 08:42:49+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.43-1.el9 started.
bachelor_project_db  | '/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'
```

ILLUSTRATION 12.2 – Commande et sortie de docker compose up dans le terminal.

Containers [Give feedback](#)View all your running containers and applications. [Learn more](#)

Container CPU usage		Container memory usage		Show charts	
101.14% / 800% (8 CPUs available)		2.71GB / 3.73GB			
<input type="text"/> Search		<input type="checkbox"/>	Only show running containers		
	Name	Container ID	Image	Port(s)	CPU (%) Last started Actions
<input type="checkbox"/>	bachelor-project	-	-	-	101.14% 59 minutes ago
<input type="checkbox"/>	bachelor_project_app_dev	6f6821345278	bachelor-project-app-dev	5001:5001	0% 11 hours ago
<input type="checkbox"/>	bachelor_project_db	3bf61d92c440	mysql:8.0	3306:3306 ↗	0.75% 59 minutes ago
<input type="checkbox"/>	bachelor_project_app	50305d547768	bachelor-project-app	5001:5001 ↗	100.39% 59 minutes ago

ILLUSTRATION 12.3 – Docker Desktop montrant les conteneurs en cours d'exécution.

12.7. VARIABLES D'ENVIRONNEMENT

Les principales variables s'ajustent dans `docker-compose.yml` (ou via `.env`).

TABLEAU 12.1 – Variables d'environnement pour la personnalisation

Variable	Rôle	Valeur par défaut
DATABASE_URI	Connexion MySQL (stops)	<code>mysql+pymysql://stops_-user:1234@db/stops_db</code>
AUTH_DATABASE_URI	Base d'authentification	<code>mysql+pymysql://.../auth_db</code>
AUTH_DB_USER,	Compte dédié auth	<i>vide par défaut</i>
AUTH_DB_PASSWORD	(optionnel)	
AUTO_MIGRATE	Init/migrate auto Alembic ²⁶	<code>true (dev)</code>
SKIP_DATA_IMPORT	Sauter pipeline de données	<code>false (app), true (app-dev)</code>
MATCH_ONLY	N'exécuter que matching + import	<code>false</code>
FLASK_ENV,	Mode Flask	<code>development, 1</code>
FLASK_DEBUG		

12.8. PERSISTANCE ET POIDS D'IMAGE

Persistante MySQL. Le volume `mysql_data` conserve la base entre redémarrages. Vous pouvez purger pour repartir de zéro :

Commande

```
1 docker compose down -v # ATTENTION: supprime le volume mysql_data
```

Contexte de build réduit. Le `.dockereignore` exclut la documentation, les gros dossiers

de données, les caches Python et métadonnées VCS (tout en gardant quelques fichiers essentiels pour MATCH_ONLY). Cela accélère docker build et évite des images obèses.

Extraits — .dockerignore

```

1 # Métadonnées VCS et IDE
2 .git
3 .vscode/
4 .idea/
5
6 # Caches Python et artefacts de build
7 __pycache__/
8 *.py[cod]
9 *.egg-info/
10
11 # Gros dossiers de données (exclus par défaut)
12 data/raw/
13 data/processed/*
14 !data/processed/osm_nodes_with_routes.csv
15 !data/processed/atlas_routes_unified.csv
16
17 # Documentation et rapport
18 memoire/
19 documentation/

```

12.9. CONSIDÉRATIONS DE SÉCURITÉ

La conteneurisation apporte plusieurs avantages sécuritaires, mais nécessite des précautions spécifiques :

Utilisateur non-root. Le conteneur s'exécute sous l'utilisateur app (UID/GID configurables), réduisant les risques d'escalade de priviléges. Les permissions des fichiers sont minimisées (644 pour les fichiers, 755 pour les répertoires).

Gestion des secrets. Les mots de passe MySQL sont passés via variables d'environnement. En production, il serait préférable d'utiliser Docker secrets³¹ ou un gestionnaire externe (HashiCorp Vault, AWS Secrets Manager).

Isolation réseau. Les conteneurs communiquent via un réseau Docker privé. Seuls les ports explicitement exposés (3306, 5001) sont accessibles depuis l'hôte.

Surface d'attaque réduite. L'image slim-bookworm contient moins de packages que l'image complète python:3.9. Le .dockerignore évite d'inclure des fichiers sensibles dans le contexte de build.

³¹. Docker. Docker secrets [en ligne]. Disponible sur : <https://docs.docker.com/engine/swarm/secrets/> (consulté le 2025-08-17).

Conclusion. Cette dockerisation est volontairement pragmatique : peu de magie, des scripts explicites, et des profils qui servent les usages quotidiens. L'approche privilégie la lisibilité et la maintenabilité, tout en respectant les bonnes pratiques de sécurité (utilisateur non-root, isolation réseau, surface d'attaque minimale). Cette base solide facilite tant le développement local que le déploiement automatisé, et peut évoluer vers des architectures plus complexes selon les besoins.

CONCLUSION

SYNTÈSE DU TRAVAIL ACCOMPLI

La digitalisation croissante des services de mobilité a rendu la qualité et la cohérence des données de transport public plus cruciales que jamais. Ce projet est né du constat d'un **désalignement** persistant entre la base de données officielle des arrêts en Suisse, ATLAS, et la référence cartographique collaborative mondiale, OpenStreetMap (OSM). Ces divergences, qu'elles soient de nature géographique, nominative ou structurelle, engendrent des incohérences qui dégradent l'expérience des usagers et complexifient les opérations pour les planificateurs.

Après filtrage, **55 571** arrêts ATLAS ont été retenus. Pour répondre à cette problématique, nous avons conçu et mis en œuvre une solution complète en trois phases. La première phase a consisté à développer un **pipeline de traitement automatisé** en Python, appliquant une cascade de méthodes d'appariement : correspondance exacte par identifiants, par nom, par proximité géographique, et enfin par analyse des lignes de transport (GTFS et HRDF).

Les résultats quantitatifs sont significatifs : nous avons réussi à établir **48 419** correspondances, ce qui représente une couverture de **84,3%** des arrêts ATLAS distincts. Ce processus a également permis de cataloguer et de prioriser des milliers d'anomalies, incluant plus de 10 000 problèmes de distance et près de 15 000 conflits d'attributs. Les tableaux suivants détaillent (i) la répartition des correspondances par méthode et (ii) les problèmes de **priorité maximale (P1)**.

CORRESPONDANCES PAR MÉTHODE

TABLEAU 12.2 – Correspondances par méthode (sur **48 419** correspondances)

Méthode	Nombre	Part (%)
Correspondances exactes	21 237	43,9
Correspondances par nom	537	1,1
Correspondances par distance	18 618	38,5
Correspondances par routes	7 021	14,5
Consolidation post-traitement	937	1,9
Propagation des duplicitas	69	0,1
Total	48 419	100,0

Détail des correspondances par distance : Étape 1 (groupe–proximité)= 15 174 ; Étape 2 (référence locale)= 129 ; Étape 3a (candidat unique)= 2 123 ; Étape 3b (ratio de distance)= 1 192.

PROBLÈMES DE PRIORITÉ MAXIMALE (P1)

La deuxième phase a porté sur le développement d'une **application web full-stack** (Flask, SQLAlchemy, JavaScript) pour la validation humaine. Cette plateforme offre une interface cartographique interactive pour visualiser les données, un outil dédié pour trier et résoudre les problèmes détectés, et

TABLEAU 12.3 – Problèmes de priorité P1 par type

Type de problème	P1 (nombre)
Distance	1 171
Non-appariements	8 192
Attributs	7 045
Total P1	16 408

un mécanisme de **persistence des solutions**, assurant que les corrections manuelles sont conservées et réappliquées lors des futurs imports de données.

Enfin, la troisième phase a consisté à **sécuriser et conteneuriser** l’application. Nous avons implémenté un système d’authentification robuste (mots de passe hachés avec Argon2, authentification à deux facteurs, protection CSRF, limitation de débit) et déployé l’ensemble de la pile logicielle (backend, base de données MySQL, frontend) avec Docker, garantissant un déploiement reproductible et simplifié.

Au moment du dépôt, voici un résumé concis des principaux types de fichiers du dépôt, incluant les données, le pipeline d’appariement, l’application web et les scripts utilisés pour le rapport.

TABLEAU 12.4 – Résumé des lignes de code par type de fichier

Type de fichier	Fichiers	Total	Non vides	Commentaires	Code
Python	70	15 006	12 755	3 051	9 705
HTML	23	1 667	1 551	73	1 478
JavaScript	17	7 519	6 706	771	5 935
CSS	35	1 681	1 428	89	1 339
TOTAL	145	25 873	22 440	3 984	18 457

Le code Python, représentant **9 705 lignes** réparties sur 70 fichiers, constitue l’épine dorsale du projet. Comme illustré dans la figure 12.4, ce code présente deux aspects complémentaires : d’une part la **répartition du nombre de fichiers** par catégorie (visualisation de gauche), et d’autre part la **distribution des lignes de code** par catégorie (visualisation de droite).

Cette double perspective révèle des insights intéressants : alors que certaines catégories contiennent de nombreux fichiers relativement courts (comme les scripts du mémoire avec 24 fichiers), d’autres concentrent beaucoup de code dans peu de fichiers (comme les blueprints Flask avec seulement 6 fichiers mais 2 173 lignes de code).

Le code se répartit principalement en trois catégories majeures :

- **Scripts du mémoire (26,1% - 2 534 lignes)** : scripts d’analyse et de visualisation utilisés pour la rédaction de ce rapport, incluant les statistiques d’appariement, les analyses de distribution de distance et la détection de problèmes.
- **Backend de l’application web (29,4% du total)** :
 - Blueprints Flask (22,4% - 2 173 lignes) pour les endpoints API
 - Services backend (3,3% - 323 lignes) pour l'email, la cryptographie et l'audit
 - Noyau backend (3,7% - 361 lignes) pour la structure principale de l’application Flask
- **Algorithmes d’appariement (19,7% - 1 912 lignes)** : le cœur du système de correspondance

des données de transport, incluant l'appariement par distance spatiale, la détection de problèmes et la logique d'appariement unifiée des routes.

Le reste du code se répartit entre le traitement des données (9,7%), l'acquisition de données (6,3%), l'infrastructure de base de données (3,7%), et divers utilitaires d'analyse (5,1%).

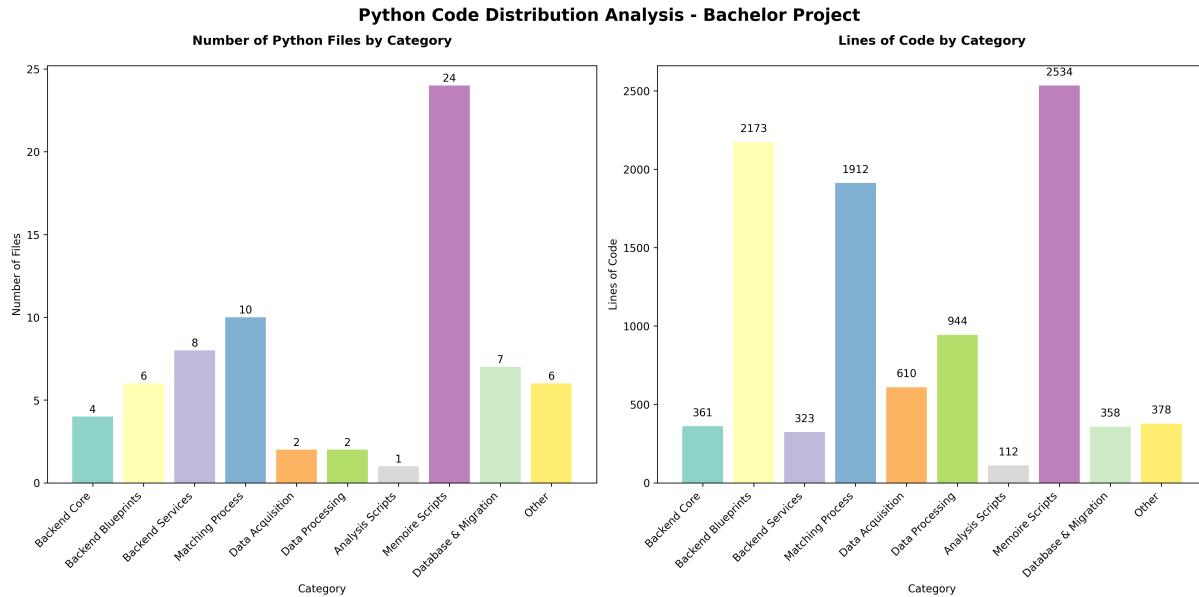


ILLUSTRATION 12.4 – Distribution du code Python : nombre de fichiers par catégorie (gauche) et lignes de code par catégorie (droite)

BILAN PERSONNEL ET RÉFLEXIF

Au-delà de ses objectifs techniques, ce projet a constitué une expérience d'apprentissage exceptionnellement riche et transversale. Sur le plan technique, il m'a permis d'acquérir et de consolider des compétences dans des domaines variés : du traitement des données à la conception d'algorithmes, en passant par le développement d'une application web complète, la sécurisation des API, la conception et l'implémentation d'un système d'authentification, ainsi que le déploiement par conteneurisation. J'ai également acquis des compétences de domaine, en me familiarisant avec différents jeux de données du monde du transport public (GTFS, HRDF). J'ai beaucoup appris sur OpenStreetMap, des connaissances utiles pour de nombreux projets.

Le défi consistant à rendre des données brutes et hétérogènes non seulement intelligibles mais aussi exploitables via une interface utilisateur intuitive a été particulièrement stimulant. La conception de l'interface utilisateur a été l'un des aspects les plus délicats : sélectionner et hiérarchiser l'information pertinente tout en préservant la simplicité d'usage.

Sur le plan de l'organisation et de la gestion de projet, ce travail m'a confronté à la nécessité d'une démarche structurée et à l'usage efficace d'outils comme Git et les IDE. La gestion des différentes facettes du projet – analyse de données, développement logiciel, rédaction technique – a exigé une planification rigoureuse et une capacité d'adaptation face aux difficultés rencontrées, que je considère comme autant d'opportunités d'apprentissage inhérentes au métier.

Ce qui m'a le plus passionné fut sans doute la dimension tangible de ce travail. Explorer les jeux de données, visualiser les réseaux de transport sur une carte et identifier des anomalies concrètes m'a offert une nouvelle perspective sur la géographie et l'infrastructure de la Suisse. La visite des bureaux des CFF à Berne fut également une expérience très enrichissante, qui a permis de contextualiser ce projet et de valider sa pertinence auprès d'experts du domaine.

PERSPECTIVES ET AMÉLIORATIONS FUTURES

Ce travail jette les bases d'un outil puissant, mais il ouvre également la voie à de nombreuses améliorations et extensions futures. Nous pouvons les regrouper en deux axes principaux.

Le premier axe concerne la **collaboration**. La prochaine étape à court terme est d'établir un canal de communication avec la communauté OpenStreetMap suisse afin de valider et d'itérer la solution et la méthode. Parallèlement, un processus de rétroaction devrait être mis en place pour signaler les incohérences avérées à l'équipe ATLAS. Pour pérenniser le projet, il serait également pertinent de le structurer en open-source, en définissant des lignes directrices pour les contributions externes.

Le deuxième axe porte sur les **améliorations techniques**. Les algorithmes d'appariement pourraient être affinés, par exemple par un modèle de scoring (éventuellement basé sur l'apprentissage automatique) qui attribuerait un score de confiance à chaque correspondance potentielle. L'application web pourrait bénéficier de fonctionnalités avancées, telles qu'une meilleure interface de résolution de problèmes pour les résoudre en masse ou encore des tableaux pour suivre l'évolution de la qualité des données.

CONCLUSION

Pas toujours fluide, parfois en correspondance serrée, mais au final ce projet est bien allé jusqu'au terminus.

RÉFÉRENCES DOCUMENTAIRES

1. Open Data Platform Mobility Switzerland [en ligne]. Disponible sur : <https://opentransportdata.swiss/en/> (consulté le 2025-07-21).
2. Traffic-points-actual-date [en ligne]. Disponible sur : <https://data.opentransportdata.swiss/en/dataset/traffic-points-actual-date> (consulté le 2025-08-01).
3. ATLAS App SBB [en ligne]. Disponible sur : <https://atlas.app.sbb.ch> (consulté le 2025-07-23).
4. Wikipédia. Liste des codes pays UIC [en ligne]. Disponible sur : https://fr.wikipedia.org/wiki/Liste_des_codes_pays_UIC (consulté le 2025-08-02).
5. OpenStreetMap Wiki. DE :Tag :highway=bus_stop [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/DE:Tag:highway=bus_stop (consulté le 2025-07-14).
6. OpenStreetMap Wiki. Public transport [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Public_transport (consulté le 2025-07-15).
7. OpenStreetMap Wiki. FR :Key :public_transport [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/FR:Key:public_transport (consulté le 2025-07-16).
8. OpenStreetMap Wiki. Proposal :Public transport schema [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Proposal:Public_transport_schema (consulté le 2025-07-17).
9. OpenStreetMap Wiki. Transport Map [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Transport_Map (consulté le 2025-07-18).
10. OpenStreetMap Wiki. Key :local_ref [en ligne]. Disponible sur : https://wiki.openstreetmap.org/wiki/Key:local_ref (consulté le 2025-07-19).
11. Overpass Turbo [en ligne]. Disponible sur : <https://overpass-turbo.eu/> (consulté le 2025-07-20).
12. SQLAlchemy. SQLAlchemy Documentation [en ligne]. Disponible sur : <https://www.sqlalchemy.org/> (consulté le 2025-08-04).
13. Flask. Documentation [en ligne]. Disponible sur : <https://flask.palletsprojects.com/> (consulté le 2025-08-05).
14. Jinja. Jinja Documentation [en ligne]. Disponible sur : <https://jinja.palletsprojects.com/> (consulté le 2025-08-06).
15. Leaflet. Leaflet Documentation [en ligne]. Disponible sur : <https://leafletjs.com/> (consulté le 2025-08-07).
16. IETF. RFC 6238 : TOTP : Time-Based One-Time Password Algorithm [en ligne]. Disponible sur : <https://www.rfc-editor.org/rfc/rfc6238> (consulté le 2025-06-15).
17. Cloudflare. Cloudflare Turnstile [en ligne]. Disponible sur : <https://www.cloudflare.com/products/turnstile/> (consulté le 2025-06-16).

18. Argon2. Argon2 documentation [en ligne]. Disponible sur : <https://argon2-cffi.readthedocs.io/en/stable/> (consulté le 2025-06-17).
19. Cryptography. Fernet (symmetric encryption) [en ligne]. Disponible sur : <https://cryptography.io/en/latest/fernet/> (consulté le 2025-06-18).
20. Amazon Web Services. Amazon Simple Email Service (SES) [en ligne]. Disponible sur : <https://aws.amazon.com/ses/> (consulté le 2025-08-11).
21. Docker. Documentation [en ligne]. Disponible sur : <https://docs.docker.com/> (consulté le 2025-08-12).
22. wkhtmltopdf. wkhtmltopdf [en ligne]. Disponible sur : <https://wkhtmltopdf.org/> (consulté le 2025-08-13).
23. Docker. Environment variables with Compose [en ligne]. Disponible sur : <https://docs.docker.com/compose/environment-variables/envvars/> (consulté le 2025-08-14).
24. Docker. Docker Compose [en ligne]. Disponible sur : <https://docs.docker.com/compose/> (consulté le 2025-08-15).
25. MySQL. MySQL 8.0 Documentation [en ligne]. Disponible sur : <https://dev.mysql.com/doc/> (consulté le 2025-08-16).
26. Alembic. Alembic Documentation [en ligne]. Disponible sur : <https://alembic.sqlalchemy.org/> (consulté le 2025-08-09).
27. Gunicorn. Gunicorn Documentation [en ligne]. Disponible sur : <https://gunicorn.org/> (consulté le 2025-08-10).
28. Nginx. Nginx Documentation [en ligne]. Disponible sur : <https://nginx.org/en/> (consulté le 2025-08-11).
29. Docker. Dockerfile HEALTHCHECK [en ligne]. Disponible sur : <https://docs.docker.com/engine/reference/builder/#healthcheck> (consulté le 2025-08-12).
30. Open Transport Data Swiss. GTFS Cookbook [en ligne]. Disponible sur : <https://opentransportdata.swiss/de/cookbook/timetable-cookbook/gtfs/> (consulté le 2025-01-15).

Fin

