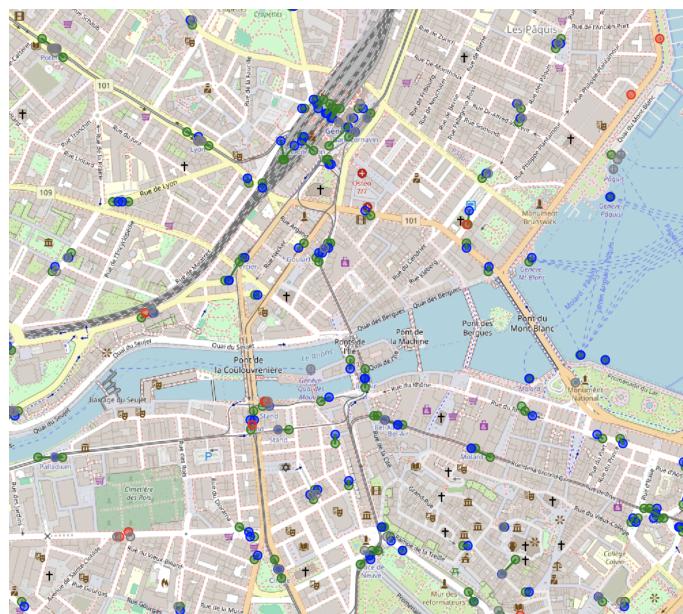


Synchronisation des Arrêts / Points d'Arrêt entre OSM et ATLAS



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	viii
Liste des illustrations	xiii
Liste des tableaux	xiv
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	31
c Étape 3 : Correspondance basée sur la proximité avec critères relatifs	31
3.5 Consolidation Post-traitement	32
3.6 Propagation des Duplicatas	32
3.7 Correspondance Manuelle	33
3.8 Résultats actuels	33

4 Chapitre 4 : Lignes et appariement par lignes	34
4.1 Des flux bruts à un fichier unique de lignes	34
a Ce qu'est <code>atlas_routes_unified.csv</code>	34
b Comment on le génère (vue d'ensemble)	35
4.2 Ce que disent les données unifiées	35
4.3 D'autres vues utiles	39
4.4 Appariement par lignes : comment ça marche	41
a Candidats par distance	41
b Paramètres	42
4.5 Appariement par lignes : les chiffres	42
a Efficacité de l'appariement par lignes : une analyse complète	42
4.6 Scripts utilisés dans ce chapitre	43
4.7 Bilan et perspectives	44
5 Chapitre 5 : Analyse des résultats	45
5.1 Comment nous avons calculé les statistiques	45
5.2 Distribution des distances par méthode	46
5.3 Par opérateur : où est-ce le plus précis ?	48
5.4 Les non-matchés et leurs « alter ego » proches	48
5.5 Autres statistiques utiles	49
5.6 Mini recettes reproductibles	49
5.7 Réflexions et pistes d'amélioration	50
6 Chapitre 6 : Détection des problèmes	51
6.1 Ce que nous détectons (types de problèmes)	51
6.2 Pourquoi prioriser	51
6.3 Comment les priorités sont attribuées	51
a Priorités pour la distance	51
b Priorités pour les non-appariements	52
c Priorités pour les attributs	53
6.4 Ce que cela donne dans l'application web	53
6.5 Chiffres de la dernière exécution	54
6.6 Persistance et flux de travail	54
6.7 Le modèle mental d'un réviseur	55
6.8 Réflexion et prochaines étapes	55
7 Chapitre 7 : Base de données et données persistantes	61
7.1 Importer les données : le rôle de <code>import_data_db.py</code>	61
7.2 Schéma logique : les tables qui comptent	62
7.3 Pros et cons du schéma vis-à-vis du rendu cartographique	64
7.4 Données persistantes : comment elles survivent aux ré-imports	65
7.5 Performance : pourquoi ça défile vite sur la carte	66
8 Chapitre 8 : Backend	67
8.1 Architecture générale	67
8.2 Blueprints et endpoints	69
a Données : <code>/api/data</code>	69
b Recherche et matches manuels	69

c	Statistiques : /api/global_stats	70
d	Rapports : /api/generate_report	71
e	Problèmes et persistance	73
8.3	Requêtage partagé et optimisation	74
8.4	Diagrammes de flux	77
8.5	Bonnes pratiques et lisibilité	78
9	Chapitre 9 : Frontend	79
9.1	Pages et navigation	79
9.2	Carte interactive (Index)	80
a	Cycle de vie des requêtes	80
b	Rendu des marqueurs et des lignes	81
c	Info-bulles et chargement paresseux	81
d	Panneau de filtres et recherche	81
9.3	Outil Problèmes	82
9.4	Données persistantes	83
9.5	Rapports et instantanés	83
9.6	Liaison avec le backend (rappel Chap. 8)	84
10	Chapitre 10 : Système d'authentification sécurisé	87
10.1	Objectifs	87
10.2	Aperçu de l'architecture	87
a	Composants	87
10.3	Modèle de données (auth_db)	88
a	Parcours utilisateur et impact sur les données	89
10.4	Sécurité opérationnelle (en pratique)	93
11	Chapitre 11 : Évaluation de la sécurité de l'application	94
11.1	Périmètre et surface d'attaque	94
11.2	Contrôles en place (aperçu technique)	95
11.3	Scénarios d'attaque et déroulé	99
11.4	Calibrage et limites actuelles	101
11.5	Améliorations et vecteurs non encore couverts	101
11.6	Conclusion	102
12	Déploiement et Conteneurisation	103
12.1	Introduction à Docker	103
12.2	Pourquoi dockeriser?	103
12.3	Les quatre pièces du puzzle	103
12.4	Aperçu	104
12.5	L'orchestration : docker-compose.yml	104
12.6	L'image : dockerfile	105
12.7	Le chef d'orchestre : entrypoint.sh	106
12.8	Démarrer en 30 secondes	107
12.9	Captures d'écran	108
12.10	Personnaliser par variables d'environnement	109
12.11	Persistance et poids d'image	109
12.12	Recettes de diagnostic	110

12.13 Réflexion	110
Conclusion	112

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

RÉSUMÉ

Ce projet a pour objectif de synchroniser les arrêts de transport public présents dans OpenStreetMap avec ceux du système officiel suisse ATLAS, afin d'améliorer la précision et la fiabilité des données. Dans un premier temps, nous analysons la structure, la couverture et les balises associées des deux jeux de données, ATLAS et OpenStreetMap, spécifiquement en Suisse.

Le cœur de notre démarche repose sur un processus de correspondance sophistiqué combinant plusieurs méthodes : correspondance exacte, par nom, par distance, par itinéraire, etc. Cette approche permet d'identifier avec précision les arrêts communs aux deux bases de données. Une première analyse statistique des résultats obtenus est ensuite réalisée.

Face aux nombreux cas problématiques rencontrés, nous avons développé une application web conviviale permettant de visualiser simultanément les deux ensembles de données ainsi que leurs correspondances. Cet outil facilite l'identification des incohérences, la génération de rapports détaillés et l'exécution d'ajustements manuels pour corriger les divergences.

Le projet se conclura par la présentation des résultats finaux, définissant une stratégie claire pour atteindre une synchronisation complète entre OpenStreetMap et ATLAS. Cela garantira un ensemble de données de transport public cohérent, fiable et exploitable facilement pour divers usages.

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

ACRONYMS

- 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	29
3.4	Correspondances – Zürich HB (étape 2)	31
3.5	Correspondance par distance – étape 3	32
4.1	Nœuds OSM hors lignes vs. sur des lignes (région de Genève).	36
4.2	OSM : tracé des lignes à partir des relations de type <code>route</code> (Genève). Géométrie bien structurée.	37
4.3	Atlas-GTFS : approximation des trajets (Genève). Lignes reconstruites depuis l'ordre des arrêts — plus fragmenté.	37
4.4	Distribution du nombre de lignes GTFS uniques par <code>sloid</code> (coupée à 20).	39
4.5	Distribution du nombre de couples (ligne, direction) par <code>sloid</code> (coupée à 30).	39
4.6	Distribution du nombre de lignes HRDF uniques par <code>sloid</code> (coupee a 20).	40
4.7	Nombre de tokens GTFS (<i>ligne, direction</i>) par <code>sloid</code> retrouvés dans OSM (coupée à 30).	43
5.1	Distances par méthode (≤ 200 m)	46

5.2	Boîtes par méthode	46
5.3	Distribution globale (log)	47
5.4	Distances par opérateur	48
5.5	Tranches de distance	49
5.6	Par type de nœud OSM	50
6.1	Filtres de l'application web : par type de problème et priorité.	57
6.2	Répartition des problèmes par type.	58
6.3	Distribution des priorités (P1/P2/P3) par type de problème.	58
6.4	Distribution des distances pour les problèmes de type distance.	59
6.5	Non-appariements par côté (ATLAS uniquement vs OSM uniquement).	59
6.6	Top opérateurs les plus concernés par les problèmes de distance.	60
6.7	Top arrêts (noms) avec le plus de problèmes.	60
9.1	Vue générale de la page d'accueil	79
9.2	Filtre ATLAS activé — ville de Zurich	80
9.3	Bannière politique de rendu selon le niveau de zoom	81
9.4	Info-bulles	82
9.5	Panneau de filtres avec « chips » actifs	83
9.6	Résumé d'en-tête avec statistiques globales. 3 filtres actifs	83
9.7	Page Problèmes — vue d'ensemble : filtres, carte de contexte et panneau de résolution	84
9.8	Contrôles de filtrage et options de persistance — Page Problèmes	85
9.9	Panneau de résolution — actions contextuelles selon le type d'anomalie	85
9.10	Données persistantes — pour utilisateurs non administrateurs	86
9.11	Page Rapports — génération (PDF/CSV) et limite quotidienne	86
10.1	Bouton « Créer un compte » visible dans l'interface.	89
10.2	Formulaire d'inscription (email, mot de passe).	89
10.3	<code>auth_db.users : email</code> et <code>password_hash</code> après inscription.	90
10.4	Processus de vérification d'email (espace réservé).	90
10.5	<code>auth_db.users : is_admin, is_email_verified, last_verification_sent_at, is_totp_enabled.</code>	91

10.6 Page de connexion. Les limites de débit et le CAPTCHA (Turnstile) freinent les robots (voir Chap. 10).	91
10.7 Connexion réussie : alertes UI sur l'email non vérifié et la 2FA non activée.	91
10.8 auth_db.users : last_login_at, failed_login_attempts, locked_until.	91
10.9 Activation 2FA avec QR <i>otpauth://</i> et secret Base32. QR et secret encodent la même information.	92
10.10 Codes de secours : affichés une seule fois à l'activation. Stockage côté serveur : hachés Argon2 dans un JSON.	92
10.11 Étape de connexion avec saisie du code 2FA (ou d'un code de secours).	93
10.12 auth_db.users : totp_secret (chiffré), backup_codes_json (haché), created_at, updated_at.	93
11.1 Verrouillage progressif : coût croissant pour l'attaquant.	99
11.2 Échec de mot de passe et blocage temporaire affiché dans l'interface.	99
11.3 Garde 2FA : étape obligatoire après mot de passe.	99
11.4 Tentative avec code 2FA erroné.	100
11.5 Exemple d'énumération d'email via message d'erreur d'inscription.	100
11.6 Cycle de vie d'un jeton de vérification.	100
12.1 Deux conteneurs reliés sur le réseau compose ; volumes pour le code (monté) et les données MySQL (persistées).	104
12.2 Commande et sortie de docker compose up dans le terminal.	108
12.3 Docker Desktop montrant les conteneurs en cours d'exécution.	108
12.4 [À remplacer] Capture d'écran complémentaire.	109

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	30
3.2	Données OSM – Münchenstein, Hofmatt	30
4.1	Structure du fichier <code>atlas_routes_unified.csv</code>	34
6.1	Compteurs de haut niveau de la dernière exécution	54
10.1	Champs de la table <code>users</code>	88
10.2	Champs de la table <code>auth_events</code>	88
12.1	Variables d'environnement pour la personnalisation	109

INTRODUCTION

CONTEXTE ET PROBLÉMATIQUE

La digitalisation 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 trouve leur **désalignement structurel**. Deux sources décrivent la même réalité — le réseau et ses arrêts — mais avec des identifiants, des hiérarchies et des coordonnées non synchronisés. Ces écarts, parfois de quelques mètres, parfois substantiels, créent des incohérences qui affectent :

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

Nous proposons une approche systématique pour **identifier, apparter et corriger** les divergences entre ATLAS et OSM, puis **outiller** la décision humaine lorsque l'automatisation ne suffit pas.

Objectifs

- Concevoir une méthodologie robuste d'appariement ATLAS ↔ OSM (exact, nom, distance, lignes).
- Quantifier les incohérences (positions, attributs, non-appariements).
- Développer une application web de visualisation, de triage et de correction avec *persistance* des décisions.
- Générer des rapports et prioriser les problèmes.
- Fournir un déploiement reproductible (conteneurs) et un code ouvert.

APPROCHE MÉTHODOLOGIQUE

Deux phases complémentaires structurent la démarche.

Phase 1 : Traitement automatisé

Scripts Python pour importer, nettoyer, enrichir et apparter les données avec différents algorithmes.

Phase 2 : Application web interactive

Un backend Flask, une base MySQL et une interface JavaScript rendent visibles les résultats, **accélèrent la validation humaine** et permettent d'enregistrer des **solutions persistantes** réappliquées aux prochains imports.

Code source

Ce document n'affiche que des extraits ciblés. L'intégralité du code est disponible sur :

https://github.com/openTdataCH/stop_sync_osm_atlas

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

Conventions de couleur

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

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

STRUCTURE DU MÉMOIRE

Le mémoire est structuré comme un parcours 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 mène naturellement à la **détection systématique des problèmes (Chapitre 6)**, où nous classons et priorisons les cas restants qui nécessitent une intervention humaine.

Pour outiller 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 Open Transport Data Swiss.¹ Cette ressource centralise les données des transports publics en Suisse, offrant une base précieuse pour l'analyse et le développement d'applications.

Sauf indication contraire, toutes les statistiques et cartes de ce chapitre sont calculées sur le snapshot de données du 18 août 2025. À partir de cette version, un filtre géographique par polygone frontière suisse (WGS84) est appliqué en plus du code pays UIC=85, afin d'exclure précisément les points situés hors de Suisse malgré un préfixe UIC suisse.

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

Frontière suisse. Le polygone de frontière (WGS84) est chargé depuis un **GeoJSON local** dans `data/raw/switzerland_boundary_osm.geojson`; aucune récupération en ligne n'est effectuée par les scripts.

1.1. ARRÊTS

L'analyse des arrêts s'appuie sur le jeu de données `traffic-points-actual-date`,² qui recense les arrêts de transport public en Suisse avec des informations sur leur localisation et leurs caractéristiques. Ces points peuvent être visualisés sur une carte interactive via l'application web <https://atlas.app.sbb.ch/>.³

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 permettant d'identifier les lieux de transport public. Les deux premiers chiffres représentent le code du pays ; la Suisse, par exemple, utilise le code 85.⁴ 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 potentiellement utile pour établir des correspondances avec d'autres jeux de données.

Le jeu de données distingue deux types de `trafficPointElementType` : `BOARDING_AREA`

1. *Open Transport Data Swiss.* URL : <https://opentransportdata.swiss/en/> (visité le 21/07/2025).

2. *Traffic-points-actual-date.* URL : <https://data.opentransportdata.swiss/en/dataset/traffic-points-actual-date> (visité le 01/08/2025).

3. *ATLAS App SBB.* URL : <https://atlas.app.sbb.ch> (visité le 23/07/2025).

4. *Liste des codes pays UIC.* URL : https://fr.wikipedia.org/wiki/Liste_des_codes_pays_UIC (visité le 02/08/2025).

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 :

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='
wgs84East')
12            # Comptage des quais
13            boarding_platforms = df[df['trafficPointElementType'] == ' '
BOARDING_PLATFORM']
14            df.to_csv(output_path, sep=';', index=False)

```

Remarque (frontière). Le filtre géographique utilise ce polygone local ; si le fichier n'est pas présent, les scripts échouent explicitement au chargement afin de garantir la reproductibilité hors-ligne.

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.

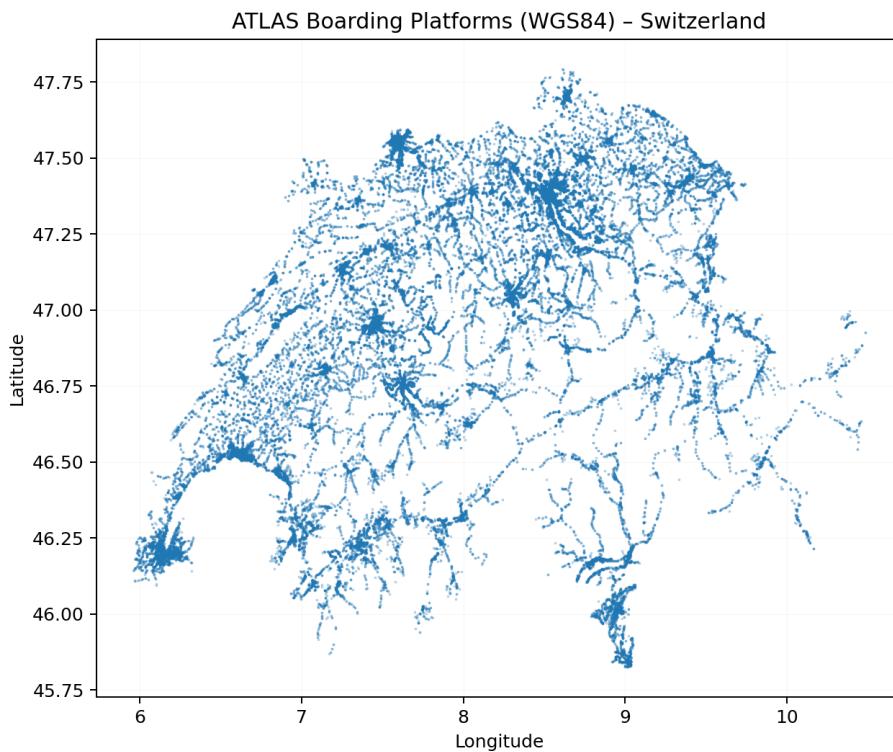


ILLUSTRATION 1.1 – ATLAS : distribution nationale (WGS84).

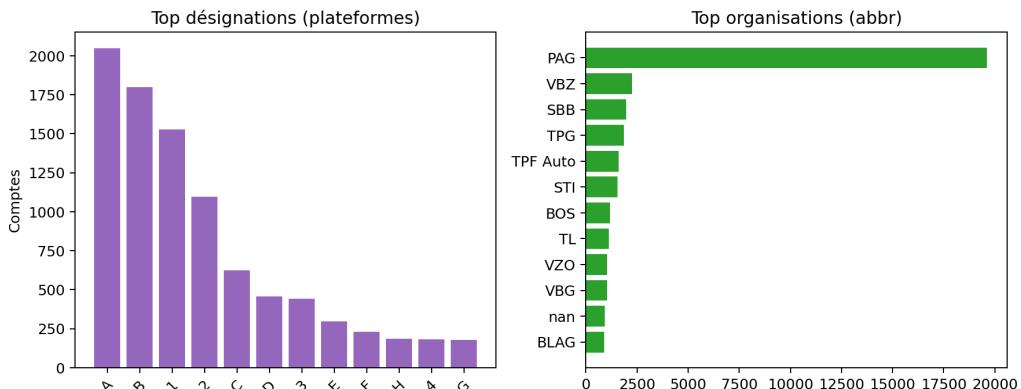


ILLUSTRATION 1.2 – À gauche : désignations de plateforme les plus fréquentes (hors valeurs manquantes). À droite : principales organisations (abrév.) déclarées.

Remarque. Les valeurs manquantes de designation (très nombreuses) sont exclues du classement pour 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 uniquement sur cet ensemble de coordonnées pour le moment.

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 plate-forme OpenTransportDataSuisse. 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 suscitent également notre intérêt. 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

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 intégré qui associe chaque arrêt aux couples (`route_id`, `direction_id`) desservis et aux noms de lignes, puis relier 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 *implémentées dans le code* sont :

- **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/plates-formes.

La mise en correspondance entre la colonne `stop_id` du fichier `stops.txt` (GTFS) et l'identifiant `sloid` d'ATLAS présente des défis significatifs. 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 *HAFAS Raw Data Format* structure des 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 Switzerland⁵.

5. <https://opentransportdata.swiss/en/cookbook/timetable-cookbook/hafas-rohdaten-format-hrdf/>

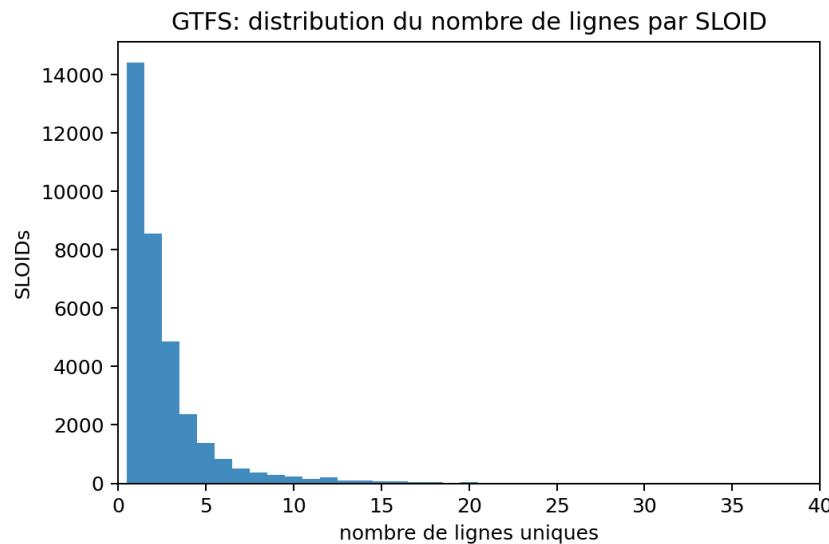


ILLUSTRATION 1.3 – GTFS : distribution du nombre de lignes par SLOID.

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.

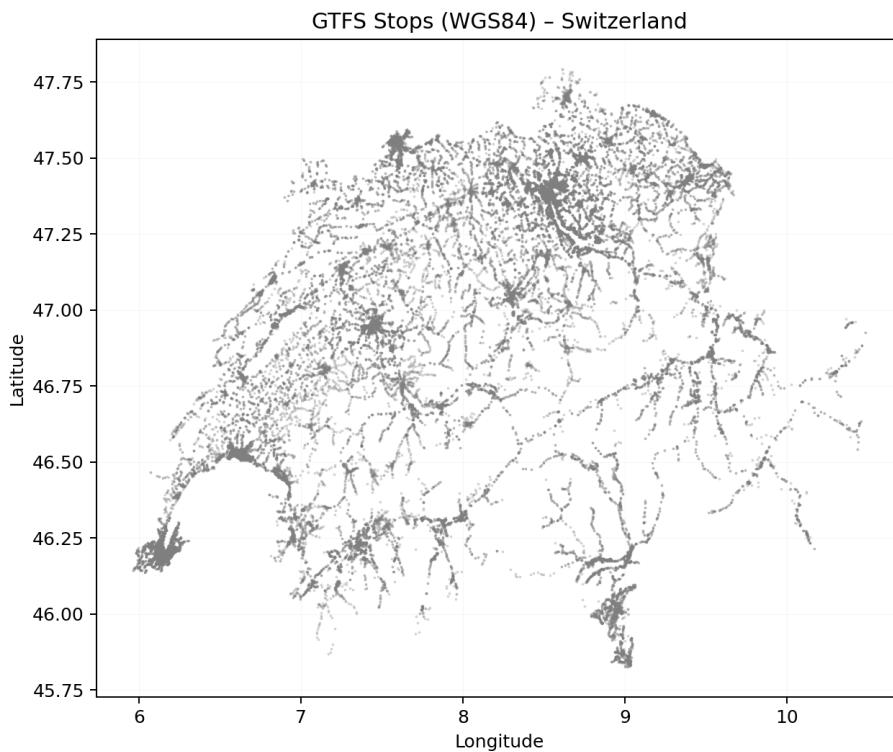


ILLUSTRATION 1.4 – Arrêts GTFS (Suisse, à l'intérieur de la frontière) : 47 567 arrêts détectés.

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.** Finalement, 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 quand un identifiant de direction explicite n'est pas disponible côté OSM.

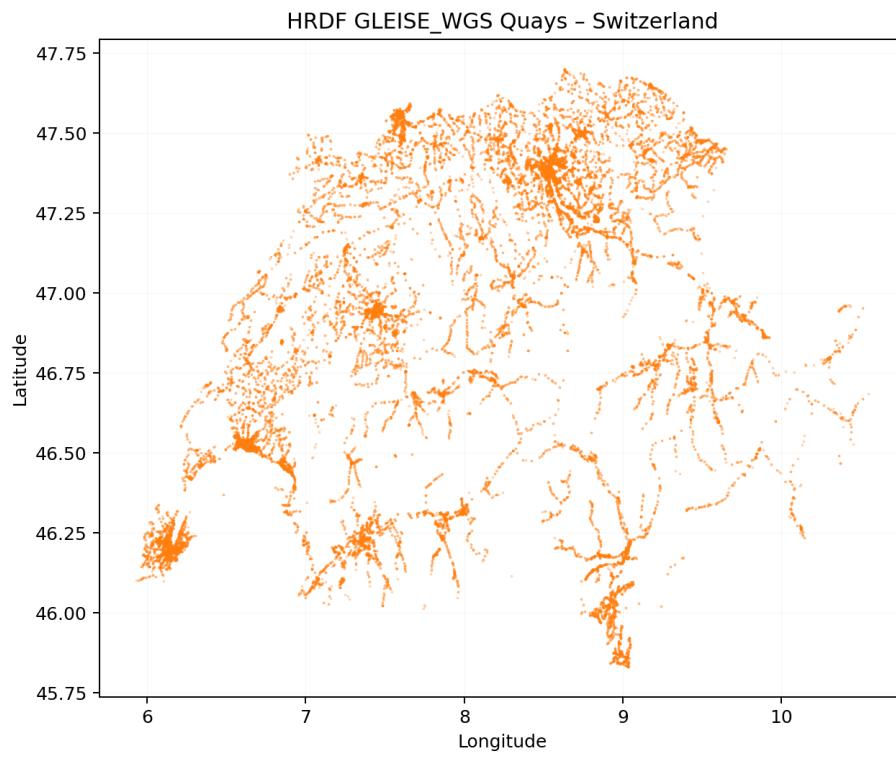


ILLUSTRATION 1.5 – Quais HRDF (GLEISE_WGS) – Suisse.

1.4. COMPARAISON GTFS ET HRDF (COUVERTURE SLOID)

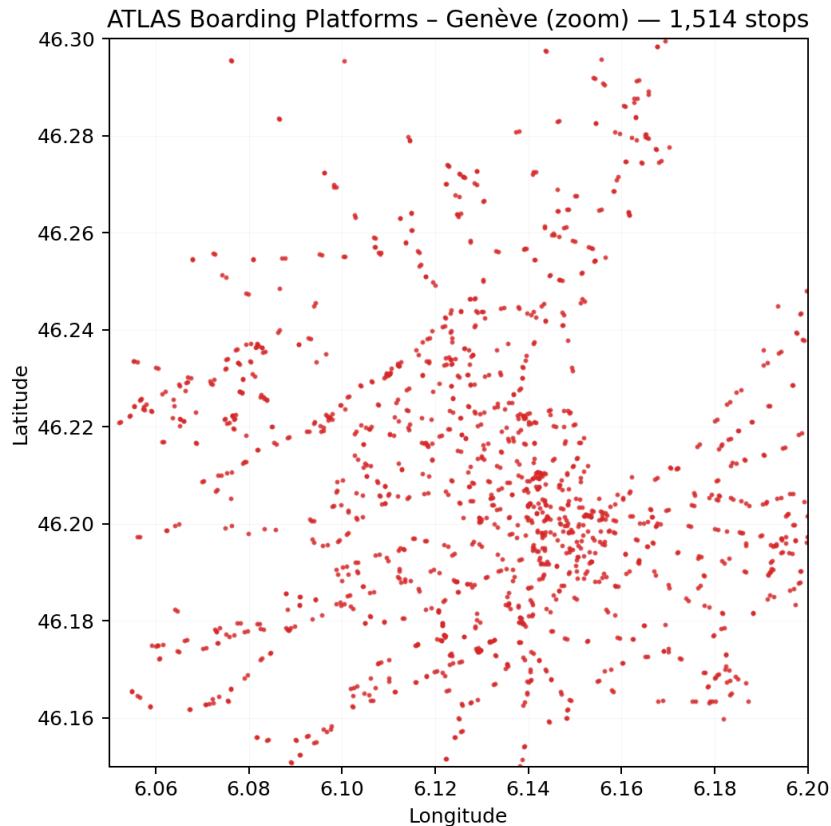


ILLUSTRATION 1.6 – ATLAS – plateformes d'embarquement, zoom Genève.

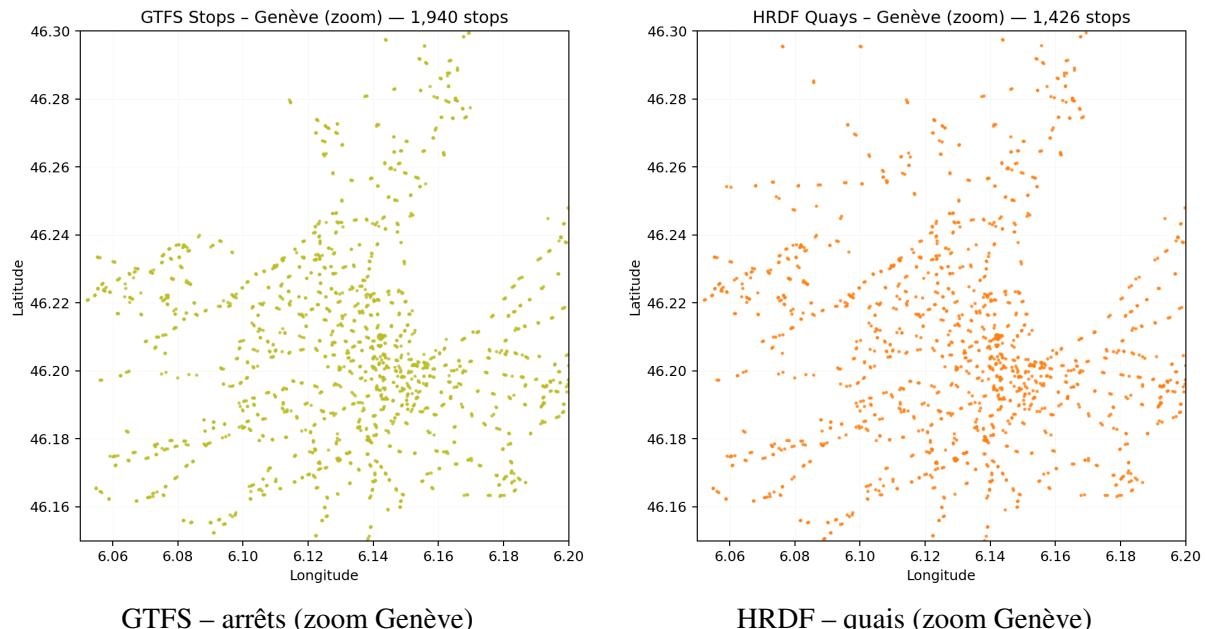


ILLUSTRATION 1.7 – Genève : comparaison des points bruts **GTFS** (gauche) et **HRDF** (droite).

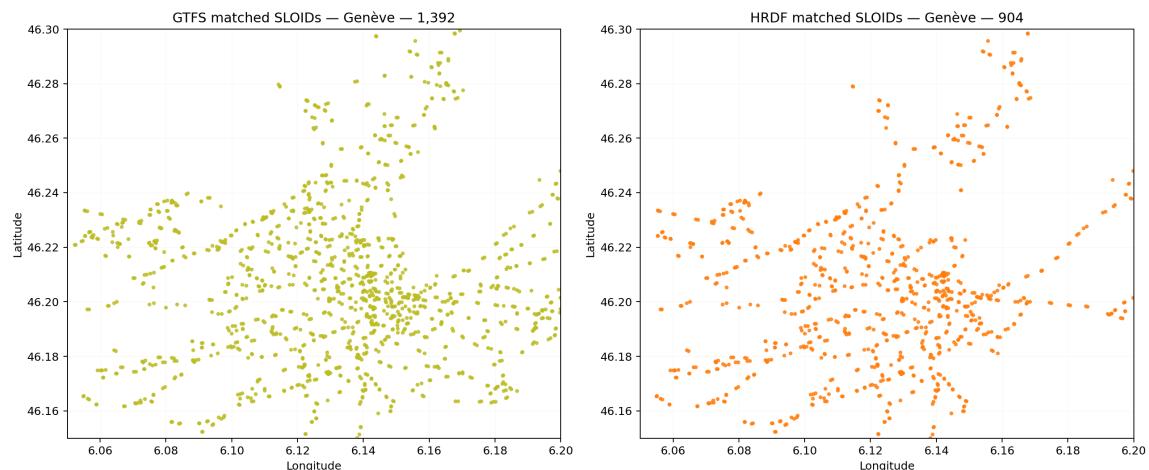


ILLUSTRATION 1.8 – Genève : SLOIDs ATLAS couverts par les jeux intégrés **GTFS** (gauche) et **HRDF** (droite).

- 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`,⁶ 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'hui.⁷ Il est important de noter qu'aucune de ces balises héritées 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'arrêt.⁸ 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

6. *DE:Tag:highway=bus_stop*. URL : https://wiki.openstreetmap.org/wiki/DE:Tag:highway=bus_stop (visité le 14/07/2025).

7. *Public transport*. URL : https://wiki.openstreetmap.org/wiki/Public_transport (visité le 15/07/2025).

8. *Public transport*, cf. note 7.

un système de balisage plus puissant mais plus complexe.⁹ L'idée est de séparer la 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),¹⁰
- 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`.¹¹ 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 traditionnels.¹²

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 sens.¹³ 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 arrêt.¹⁴ 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 ici que les **nœuds**. Par simplification, les *plateforms* ou *stop_positions* modélisés comme **zones** (*ways/relations*) ne sont pas intégrés 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 :

9. *Proposal :Public transport schema*. URL : https://wiki.openstreetmap.org/wiki/Proposal:Public_transport_schema (visité le 17/07/2025).

10. *FR :Key :public_transport*. URL : https://wiki.openstreetmap.org/wiki/FR:Key:public_transport (visité le 16/07/2025).

11. *FR :Key :public_transport*, cf. note 10.

12. *DE :Tag :highway=bus_stop*, cf. note 6.

13. *Proposal :Public transport schema*, cf. note 9.

14. *Public transport*, cf. note 7.

— ref vs local_ref (codes d’arrêt) : De nombreux arrêts de transport public possèdent un code ou identifiant officiel (numéro ou lettre fourni par l’autorité de transport). Les cartographes utilisent tantôt la balise générique `ref=`, tantôt `local_ref=`. La recommandation OSM est : 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/lettre propre à un contexte plus restreint.¹⁵

Par exemple, un arrêt de bus qui a l’ID “3154” dans la base de la ville se balisera `ref=3154`. Et 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 depuis la base de données OpenStreetMap.¹⁶ Il utilise un langage de requête appelé Overpass Query Language, qui permet de rechercher et 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 le 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

15. *Key :local,ref*. URL : https://wiki.openstreetmap.org/wiki/Key:local_ref (visité le 19/07/2025).

16. *Overpass Turbo*. URL : <https://overpass-turbo.eu/> (visité le 20/07/2025).

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 présentes sur ces nœuds. 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 le voit, 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

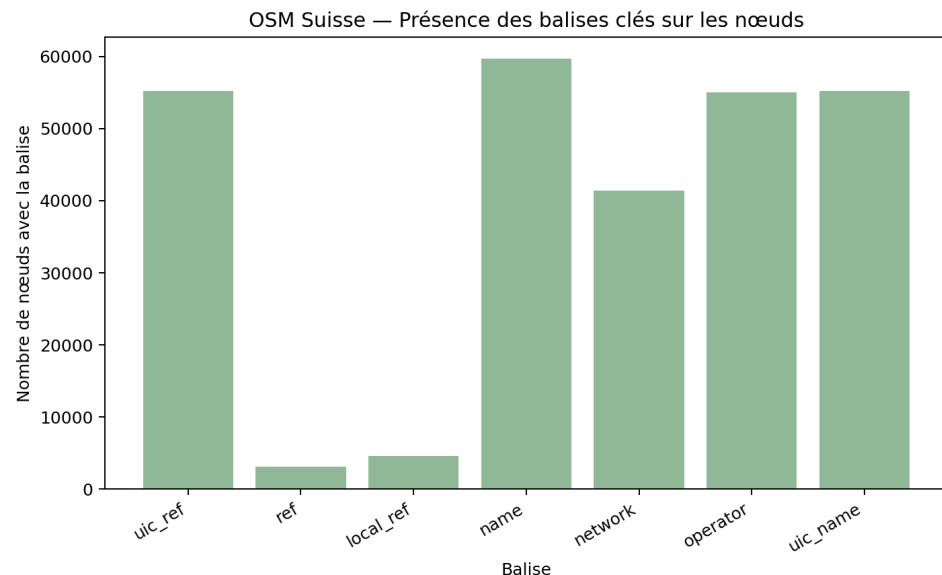


ILLUSTRATION 2.1 – Présence des balises clés par nœud (uic_ref, ref, local_ref, name, network, operator, uic_name).

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

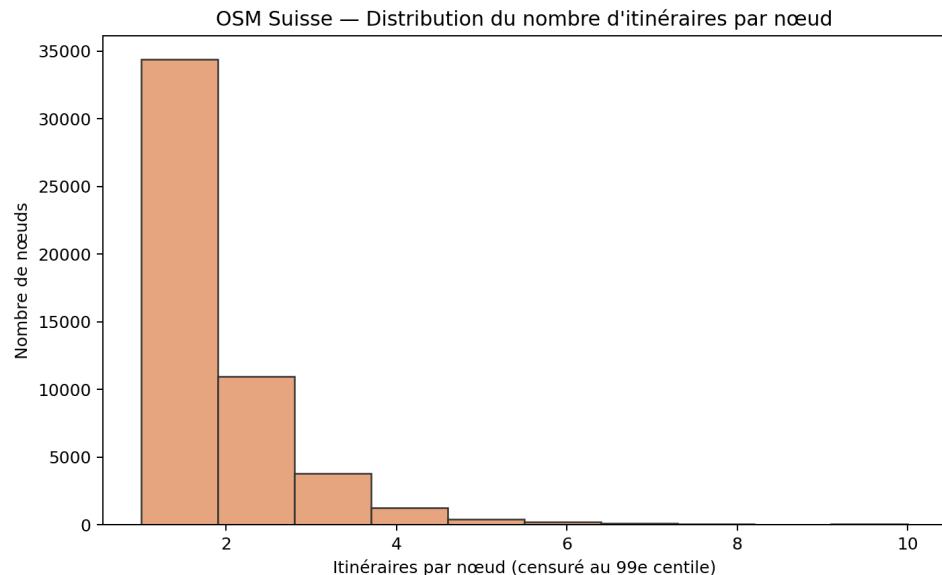


ILLUSTRATION 2.2 – OSM (Suisse) — distribution du nombre d’itinéraires distincts par nœud (censurée au 99e centile).

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 sous `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

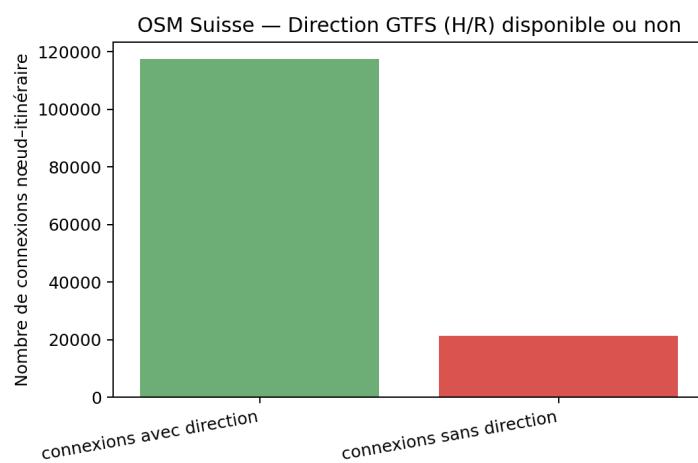


ILLUSTRATION 2.3 – Connexions noeud–itinéraire pour lesquelles une direction (H/R) est déduite à partir de ref_trips.

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 de correspondance séquentielle 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 considéré une approche alternative consistant à exécuter toutes les méthodes sur toutes les entrées, permettant 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. **Correspondance exacte** : Appariement basé sur les identifiants UIC et 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 référence 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 correspondant, 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

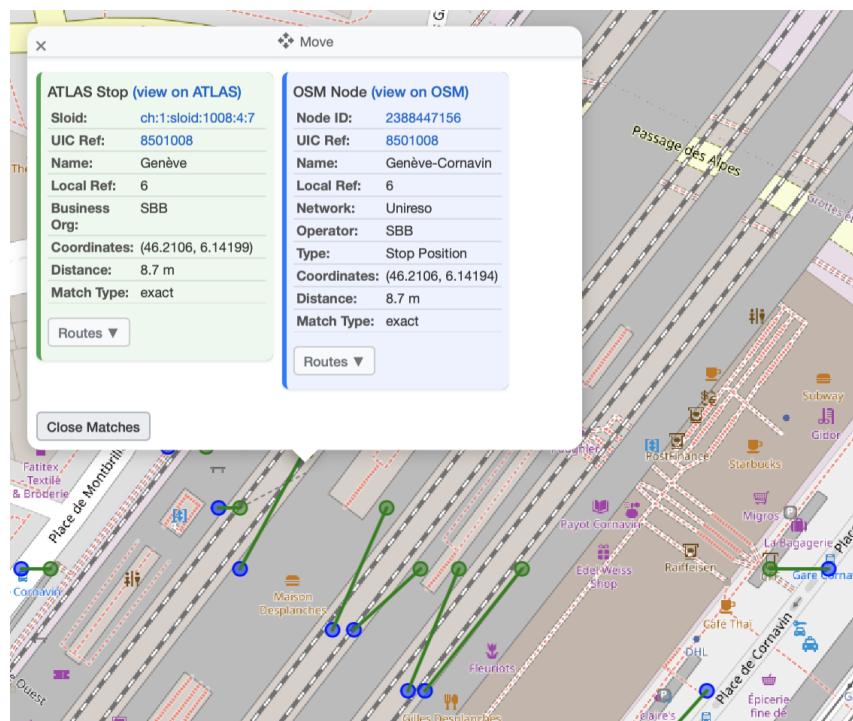


ILLUSTRATION 3.1 – Correspondances exactes à la gare de Genève-Cornavin. Les détails de l'arrêt de la voie 6 sont visibles sur l'image.

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 15174 correspondances supplémentaires.

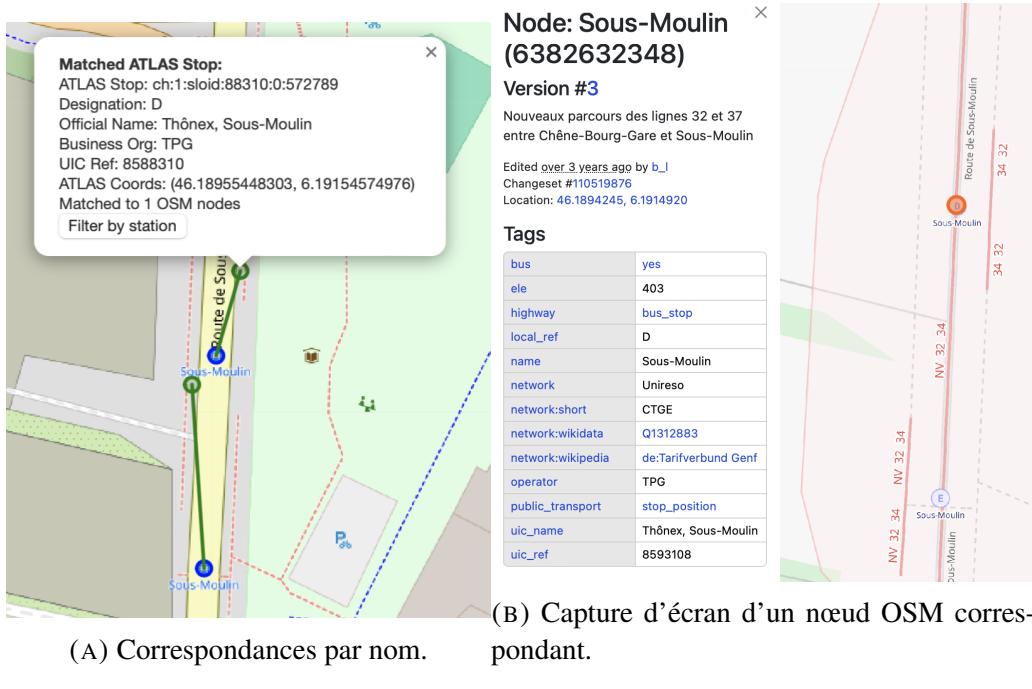


ILLUSTRATION 3.2 – Pour l'arrêt "Thônex, Sous-Moulin, D", on peut voir que, malgré une référence UIC différente, il est possible d'établir des correspondances grâce au nom.

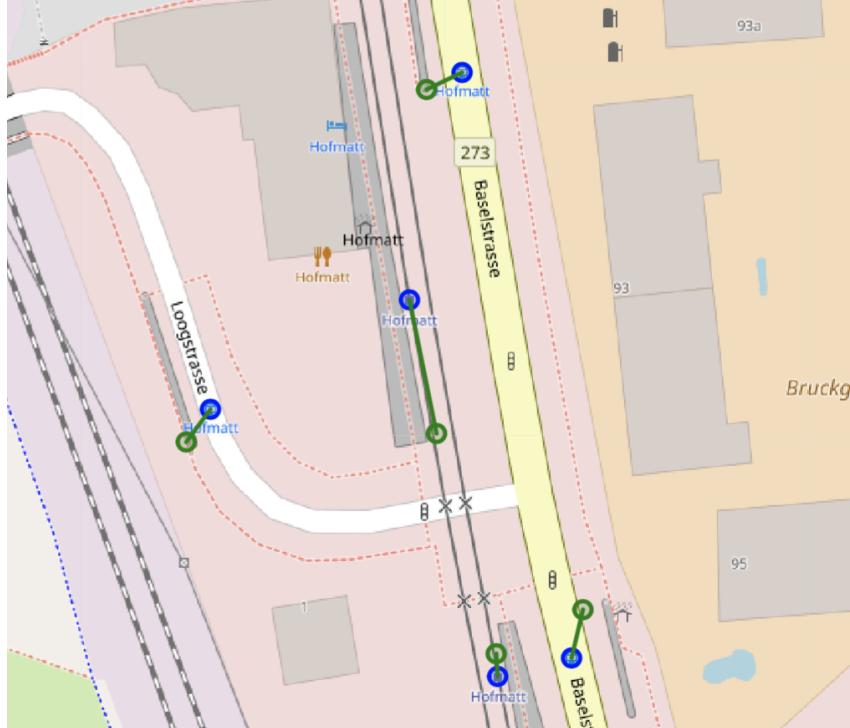


ILLUSTRATION 3.3 – Correspondances pour les arrêts de Münchenstein, Hofmatt. Malgré les divergences de uic_ref et le manque de références locales, nous avons réussi à établir des correspondances grâce à la correspondance de groupe basée sur les distances.

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.

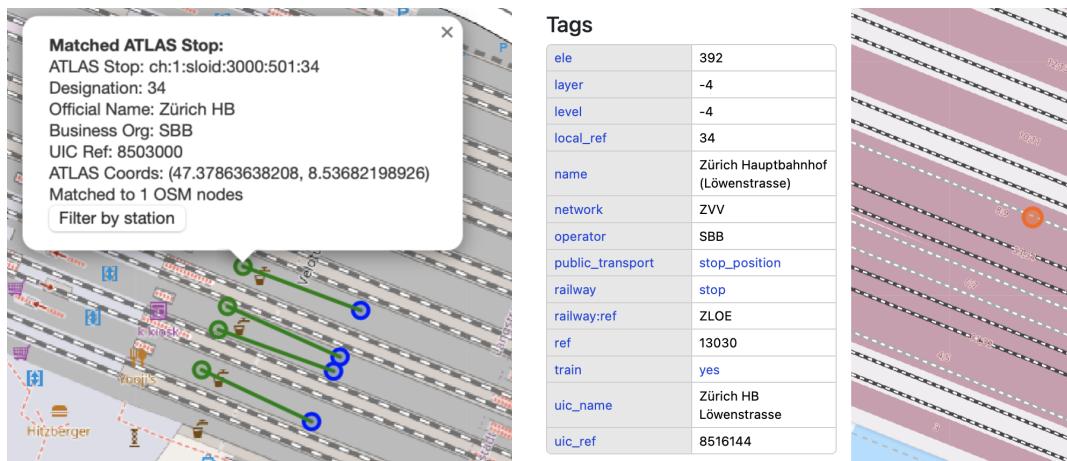


ILLUSTRATION 3.4 – Correspondances à Zürich HB grâce à l'étape 2.

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 2123 correspondances avec l'option a) et 1192 correspondances avec l'option b). Cette méthode est utile pour les cas où il y a des nœuds isolés, comme des télésièges.

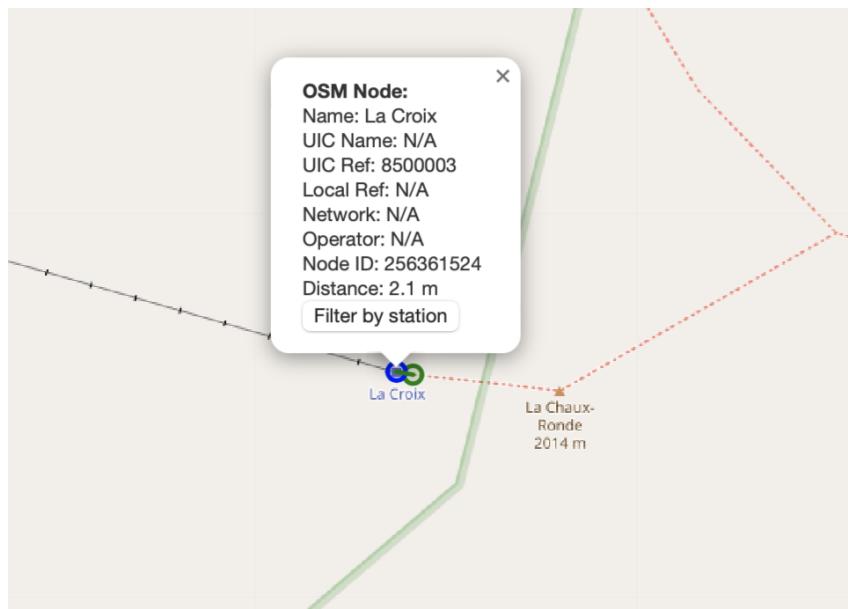


ILLUSTRATION 3.5 – Correspondance par distance étape 3 : exemple d'un arrêt isolé où un seul candidat OSM est trouvé dans le rayon de 50 mètres.

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 pour récupérer les correspondances exactes qui auraient pu être manquées lors du passage initial. Cette étape, appelée "unique-by-UIC consolidation", 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.

3.6. PROPAGATION DES DUPLICATAS

Une dernière étape consiste à propager les correspondances trouvées vers les 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 correspon-

dance 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 verra plus tard.

3.8. RÉSULTATS ACTUELS

Parmi les 55571 arrêts ATLAS que nous avons considérés, le processus de correspondance a permis d'identifier un total de 48419 correspondances entre les données ATLAS et OSM, réparties comme suit :

- Correspondances exactes : 21237
- Correspondances par nom : 537
- Correspondances par distance : 18618
 - Étape 1 (groupe-proximité) : 15174
 - Étape 2 (référence locale) : 129
 - Étape 3a (candidat unique) : 2123
 - Étape 3b (ratio de distance) : 1192
- Correspondances par routes : 7021 (détailé au chapitre 4)
- Consolidation post-traitement : 937
- Propagation des duplicates : 69

Après ces étapes, 8707 entrées ATLAS restent non appariées, et 19115 nœuds OSM restent inutilisés. Parmi ces nœuds OSM inutilisés, 14012 sont associés à au moins une route, 14917 possèdent une référence UIC, et 879 ont une référence locale (`local_ref`).

Parmi les entrées ATLAS non appariées, 3911 n'ont aucun nœud OSM dans un rayon de 50 mètres, indiquant des zones où la couverture OSM est insuffisante par rapport aux données ATLAS.

CHAPITRE 4 : LIGNES ET APPARIEMENT PAR LIGNES

Ce chapitre est 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. Attendez-vous à des extraits de code concis, des graphiques parlants et des explications pragmatiques.

4.1. DES FLUX BRUTS À UN FICHIER UNIQUE DE LIGNES

Cette section approfondit la vue d'ensemble du Chapitre 1 ; nous répétons volontairement certains points clés pour la fluidité.

a. Ce qu'est `atlas_routes_unified.csv`

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

- **GTFS** (transport public) : 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** (horaire ferroviaire) : 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 inter-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)

À haut niveau (voir `get_atlas_data.py`) :

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 `memoire/scripts_used/chap4`. Les sorties sont archivées dans `memoire/data/processed/chap4/`.

Vue GTFS

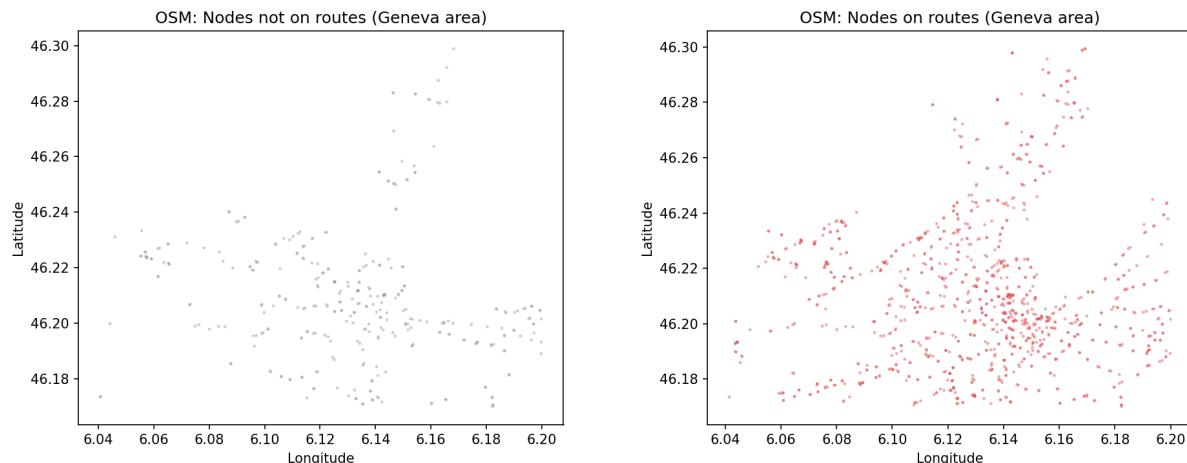
- **SLOIDs avec lignes GTFS : 34 781**
- **Nombre moyen de lignes uniques par sloid : 2,73** (médiane **2,00**)
- **Nombre moyen de couples (ligne, direction) par sloid : 4,39** (médiane **3,00**)
- **Groupes dupliqués pour (sloid, route_norm, direction) : 96,96%**
- **Même ligne+direction, plusieurs chaînes de direction : 96,95%** des groupes présentent plus d'une chaîne premier→dernier (patrons 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**)

- **Queues longues** : plusieurs couples (sloid, ligne) affichent **30–40** paires UIC premier→dernier distinctes (branches, demi-tours)

En deux mots. *GTFS* couvre un grand nombre d'arrêts avec plusieurs directions par ligne ; *HRDF* confirme une forte variété de directions terminales pour certaines lignes (queues longues). Cela implique qu'une comparaison robuste doit gérer la multiplicité des directions, pas seulement des 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).

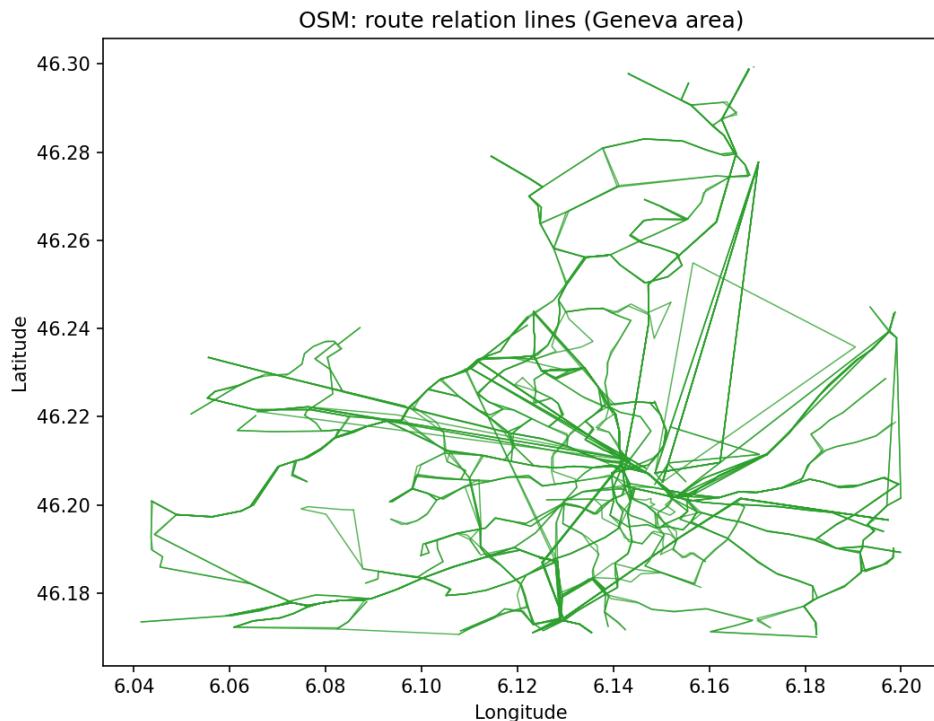


ILLUSTRATION 4.2 – OSM : tracé des lignes à partir des relations de type route (Genève). Géométrie bien structurée.

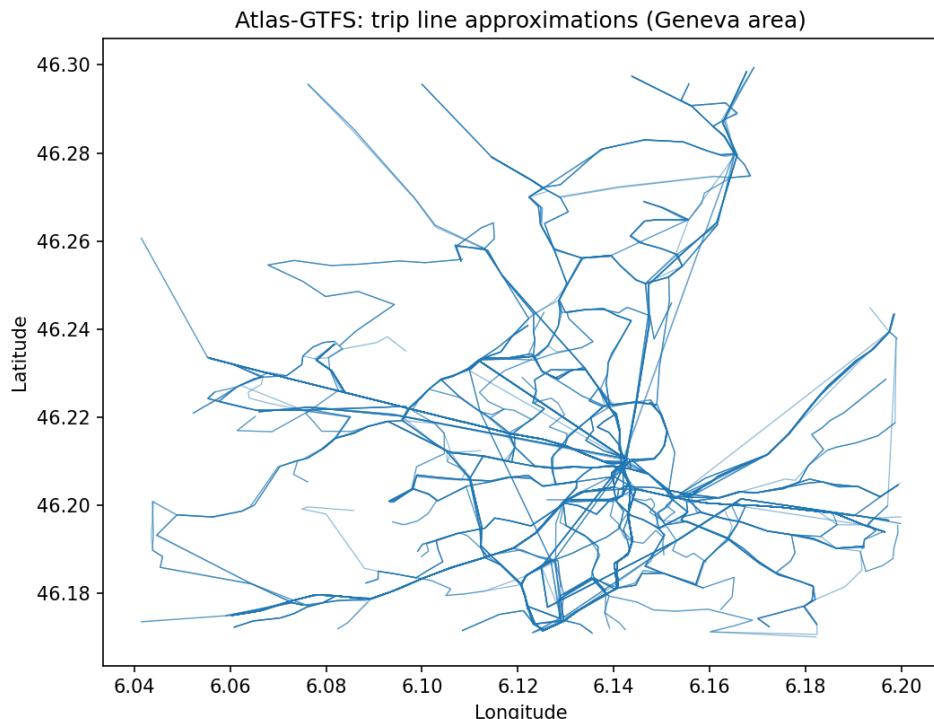


ILLUSTRATION 4.3 – Atlas-GTFS : approximation des trajets (Genève). Lignes reconstruites depuis l'ordre des arrêts — plus fragmenté.

Comment traçons-nous ces couches ?

OSM (routes) : nous parcourons les relations type=route, récupérons la séquence de membres de type node (ou way si disponible) et relions leurs coordonnées lorsque le nœud se trouve dans la boîte Genève. Cela produit un *linework* fidèle à la modélisation OSM.

Atlas-GTFS (trajets) : nous reconstruisons des polylignes en ordonnant les arrêts d'un trajet par stop_sequence et en traçant les segments entre arrêts successifs, restreints à la boîte Genève. Pour éviter le fouillis, nous priorisons les séquences répétées (plus représentatives). Extrait en pseudo-code :

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

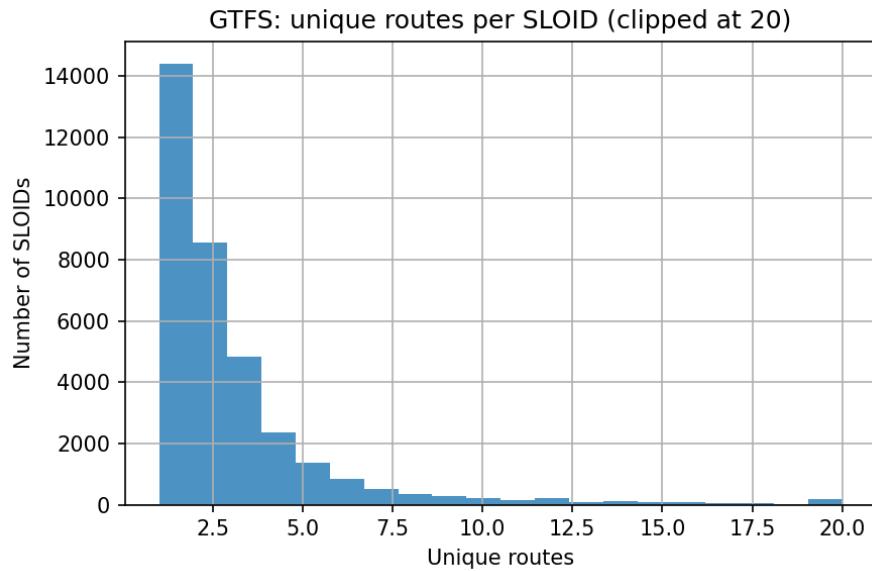


ILLUSTRATION 4.4 – Distribution du nombre de lignes GTFS uniques par sloid (coupée à 20).

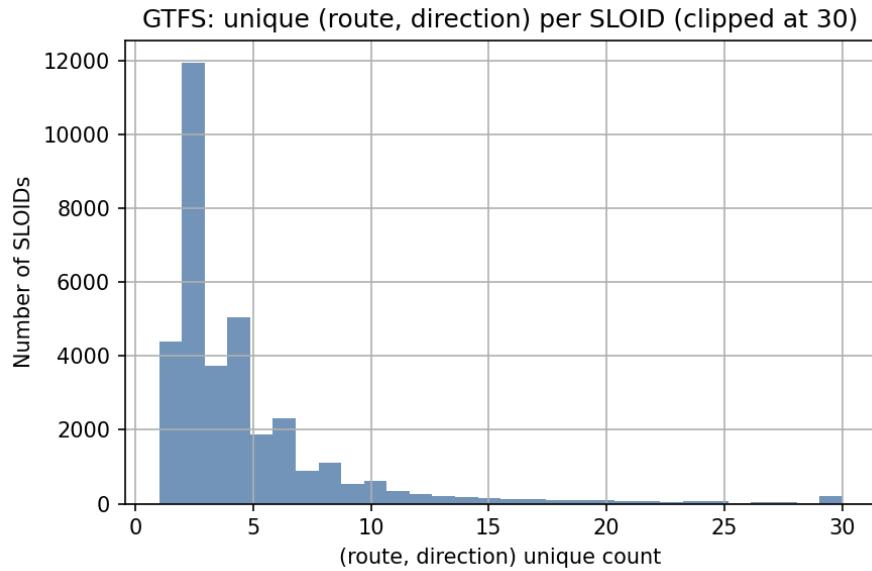


ILLUSTRATION 4.5 – Distribution du nombre de couples (ligne, direction) par sloid (coupée à 30).

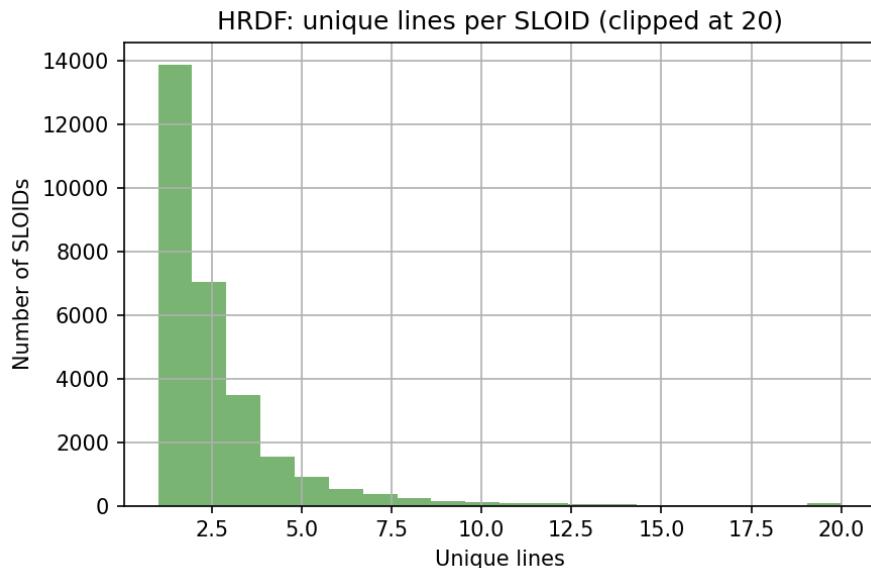


ILLUSTRATION 4.6 – Distribution du nombre de lignes HRDF uniques par sloid (coupee a 20).

- **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 (détailée 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 chaines UIC premier→dernier dérivées d'OSM pour appuyer l'appariement au niveau HRDF lorsque les identifiants GTFS

manquent dans OSM.

4.4. APPARIEMENT PAR LIGNES : COMMENT ÇA MARCHE

L'appariement compare les tokens de ligne connus pour un **sloid** ATLAS aux tokens dérivés des noeuds OSM à proximité (KD-tree, rayon configurable : 50 m par défaut). Les tokens sont soit **GTFS** (`route_id`, `direction_id`), avec normalisation éventuelle, 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. APPARIEMENT PAR LIGNES : LES CHIFFRES

Avec le script optimisé (`route_matching_stats.py`, exécuté sur les fichiers déjà présents), nous obtenons :

- **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**.

Lecture rapide : 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.

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 insights 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 \leftrightarrow nœud
- **Appariement par lignes** : **29 582** paires arrêt \leftrightarrow 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 nœuds OSM géographiquement cohérents.

Insights stratégiques. 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 nouvelle.

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** nœuds 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.

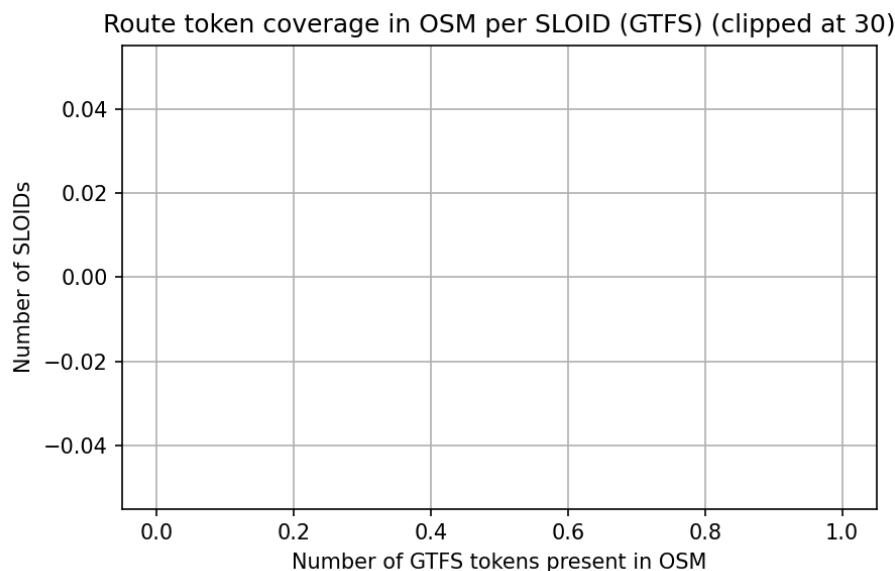


ILLUSTRATION 4.7 – Nombre de tokens GTFS (*ligne, direction*) par sloid retrouvés dans OSM (coupée à 30).

4.6. SCRIPTS UTILISÉS DANS CE CHAPITRE

Tous les scripts résident sous `memoire/scripts_used/chap4` et n'opèrent que sur `data/raw` et `data/processed` :

- `compute_unified_stats.py` : lit `atlas_routes_unified.csv`, produit des statistiques (avec impressions de distribution sur le terminal) et des histogrammes.
- `geneva_maps.py` : génère les vues OSM et ATLAS (incluant désormais les nœuds OSM hors et sur relations de ligne).
- `geneva_route_lines.py` : trace les lignes OSM, **Atlas-GTFS** et HRDF (détection robuste de FPLAN, traitement de plus d'un million de trajets).
- `route_matching_stats.py` : calcule chevauchements et couvertures, avec logs de progression et un mode rapide.
- `route_effectiveness_analysis.py` : analyse comparative complète de l'efficacité de l'appariement par lignes vs. exact.
- `compare_exact_vs_route.py` : comparaison directe des deux approches d'appariement pour quantifier la complémentarité.

4.7. BILAN ET PERSPECTIVES

Atouts. L'écriture unifiée offre une vue compacte et exploitable des lignes par arrêt, toutes sources confondues. Le pipeline d'appariement privilégie la précision (intersections de tokens) avec des seuils de distance conservateurs. La normalisation par année stabilise les comparaisons inter-versions.

En bref : l'appariement par lignes ajoute un signal fort et indépendant qui complète les méthodes exactes/nominales/distances. 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 présente une lecture approfondie — mais accessible et agréable — de la qualité de nos correspondances ATLAS ↔ OSM. Nous mêlons graphiques, petits extraits de code et indicateurs lisibles, avec une attention particulière aux cas limites et aux pistes d'amélioration.

5.1. COMMENT NOUS AVONS CALCULÉ LES STATISTIQUES

Nous utilisons directement la base MySQL du projet (variable d'environnement `DATABASE_URI`). Les scripts sont fournis et versionnés sous `memoire/scripts_used/chap5`. Voici un minuscule extrait montrant le chargement des données et le calcul de statistiques par méthode de correspondance :

Extrait Python

```
1 # memoire/scripts_used/chap5/chap5_distance_distributions.py (extrait)
2 engine = create_engine(os.getenv('DATABASE_URI', 'mysql+pymysql://...'))
3 df = pd.read_sql(
4     """
5     SELECT distance_m, match_type, osm_node_type
6     FROM stops
7     WHERE stop_type = 'matched' AND distance_m IS NOT NULL
8     """", engine)
9 summary = (
10    df.groupby('match_type')[["distance_m"]]
11    .agg(count='count', mean='mean', median='median')
12    .reset_index()
13 )
14 summary.to_csv('.../distance_summary_by_method.csv', index=False)
```

Tous les chiffres et graphiques de ce chapitre sont produits par ces scripts, exécutés sur la base actuelle.

5.2. DISTRIBUTION DES DISTANCES PAR MÉTHODE

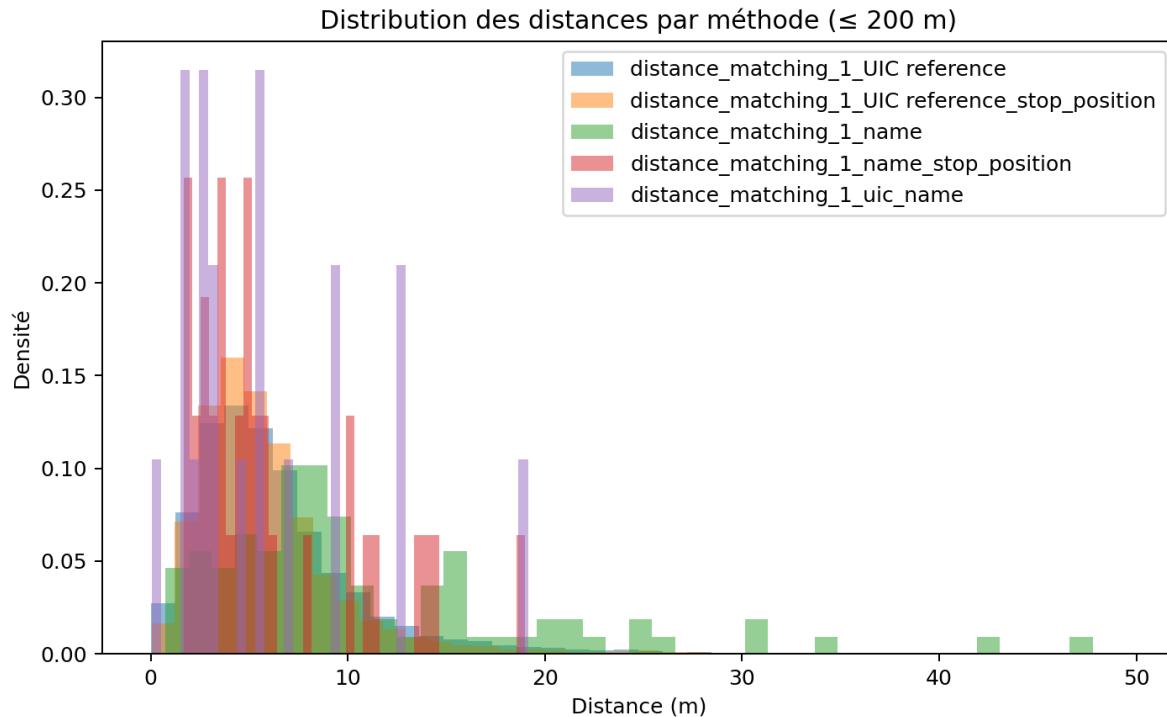


ILLUSTRATION 5.1 – Superposition des distributions des distances par méthode de correspondance (coupée à 200 m pour mieux distinguer le cœur).

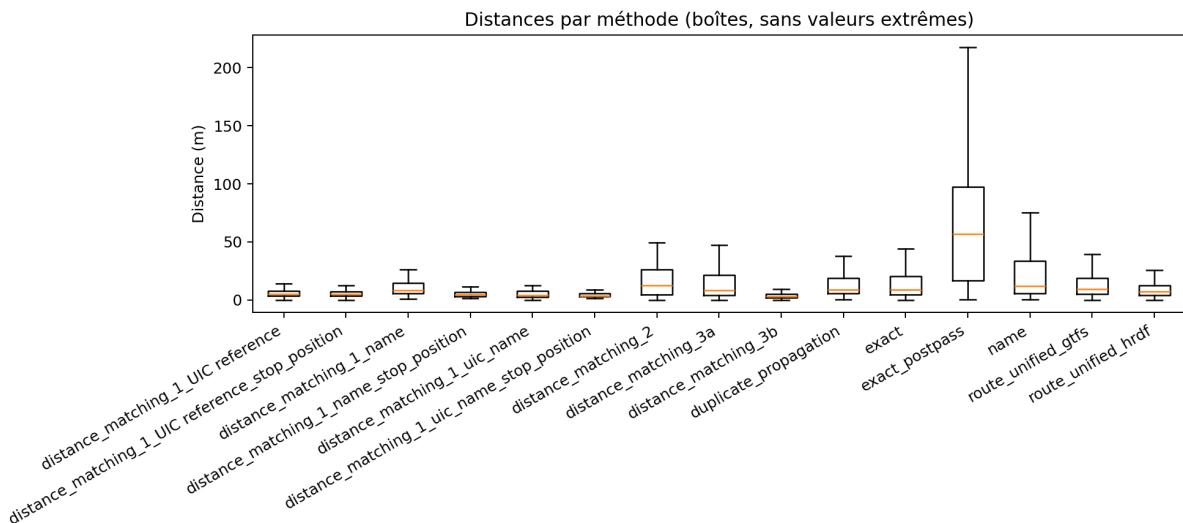


ILLUSTRATION 5.2 – Boîtes à moustaches par méthode (sans valeurs extrêmes) — lecture immédiate des médianes et de la dispersion.

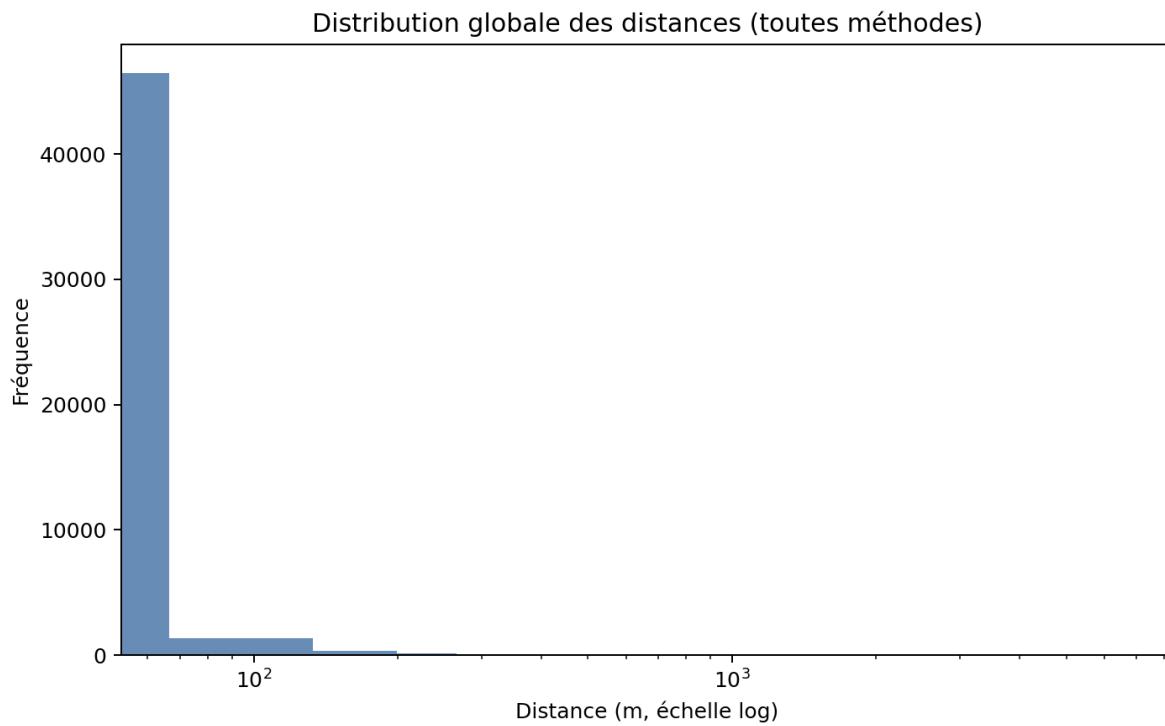


ILLUSTRATION 5.3 – Distribution globale des distances (échelle logarithmique) — utile pour visualiser la longue traîne.

Extraits des résultats par méthode (fichier `distance_summary_by_method.csv`) :

Extrait CSV

```

1 match_type,count,mean,median,p90,p95,p99,pct_≤10m,pct_≤20m,pct_≤50m,pct_≤100m
2 exact,19557,22.05,8.88,50.81,81.82,206.29,54.6%,74.8%,89.8%,96.5%
3 route_unified_gtfs,4020,13.46,9.23,32.97,40.74,47.60,53.5%,77.7%,100%,100%
4 route_unified_hrdf,2585,10.11,7.15,22.23,30.63,44.87,66.1%,87.5%,100%,100%
5 distance_matching_3b,1145,3.63,3.22,6.70,7.87,9.89,99.0%,100%,100%,100%
6 ... (voir CSV complet)

```

Points saillants :

- Les méthodes « distance 3b » et « distance (UIC/stop_position) » sont extrêmement précises (médianes \approx 3–5 m).
- Les « exacts » recouvrent des cas hétérogènes : médiane correcte (\approx 9 m) mais longue traîne liée aux homonymies et ancrages OSM imparfaits.
- Les unifications de lignes GTFS/HRDF sont globalement saines (médianes \approx 7–10 m), avec une dispersion modérée.

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

Nous joignons chaque arrêt à son opérateur ATLAS et analysons les distances des arrêts *matchés*. Nous affichons ici un classement visuel des opérateurs les plus représentés.

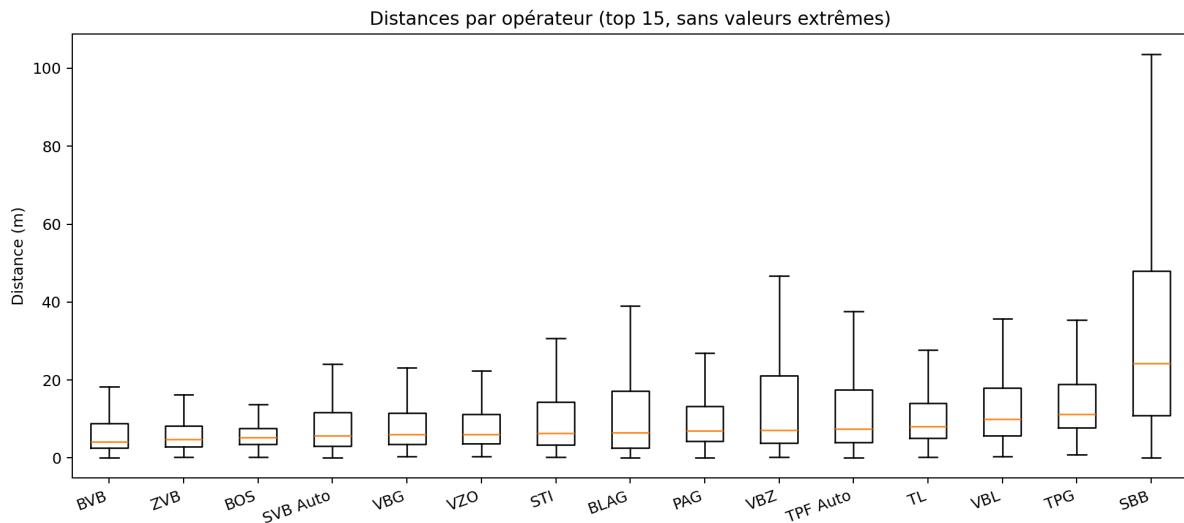


ILLUSTRATION 5.4 – Dispersion des distances par opérateur (top 15 par effectif).

Observations rapides :

- Les opérateurs urbains à forte densité de données (p. ex. TPG, VBZ, PAG) présentent des médianes de l'ordre de 7–12 m, cohérentes avec un calage cartographique fin.
- Certains opérateurs de montagne ou réseaux spéciaux montrent des dispersions plus larges — terrain complexe, géoréférencement moins standardisé.

5.4. LES NON-MATCHÉS ET LEURS « ALTER EGO » PROCHES

Combien d'entrées non-correspondantes ont pourtant un arrêt matché à proximité ? Nous échantillonons des rayons de 25, 50, 100, 200 et 400 m autour de chaque non-matché (sur coordonnées ATLAS).

Extrait CSV

```

1 radius_m,unmatched_total,with_nearby_counterpart,pct
2 25,8416,1827,21.71
3 50,8416,2572,30.56
4 100,8416,2989,35.52
5 200,8416,2992,35.55

```

Donc **entre 20% et 34%** des non-matchés ont un correspondant plausible très proche. Deux scénarios fréquents : (i) doublon/homonymie ATLAS à consolider ; (ii) ancrage OSM existant mais filtré (type de nœud, règles). Ces cas sont d'excellents candidats à la remédiation semi-automatique.

5.5. AUTRES STATISTIQUES UTILES

Répartition par tranches de distance

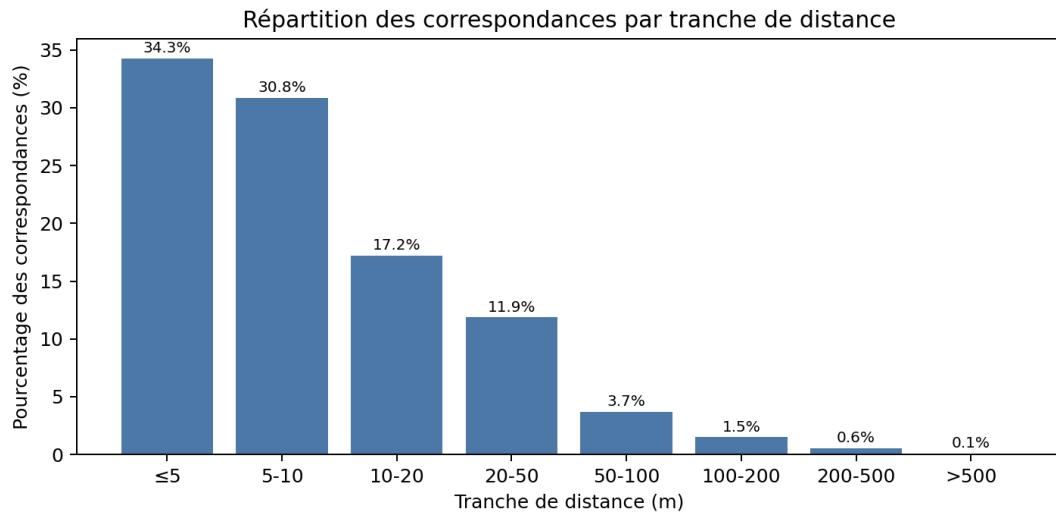


ILLUSTRATION 5.5 – Quelle part des correspondances tombent sous 5 m, 10 m, 20 m, etc.

Cette vue permet d'orienter des objectifs de qualité (p. ex. « $80\% \leq 10 \text{ m}$ »).

Surveiller la traîne longue

Nous listons également les correspondances $> 300 \text{ m}$ (`suspicious_long_distances.csv`). Cela représente 104 cas à reclasser en priorité (erreur de rattachement, homonymie distante, etc.).

5.6. MINI RECETTES REPRODUCTIBLES

Extrait Python

```

1 # Distances par type de nœud OSM (extrait)
2 subset = df[df['osm_node_type'].fillna('inconnu').isin(top_types)]
3 order = subset.groupby('osm_node_type')[ "distance_m"]\
4     .median().sort_values().index.tolist()
5 plt.boxplot([subset[subset['osm_node_type'] == t]['distance_m'] for t in order],
6             labels=order, showfliers=False)

```

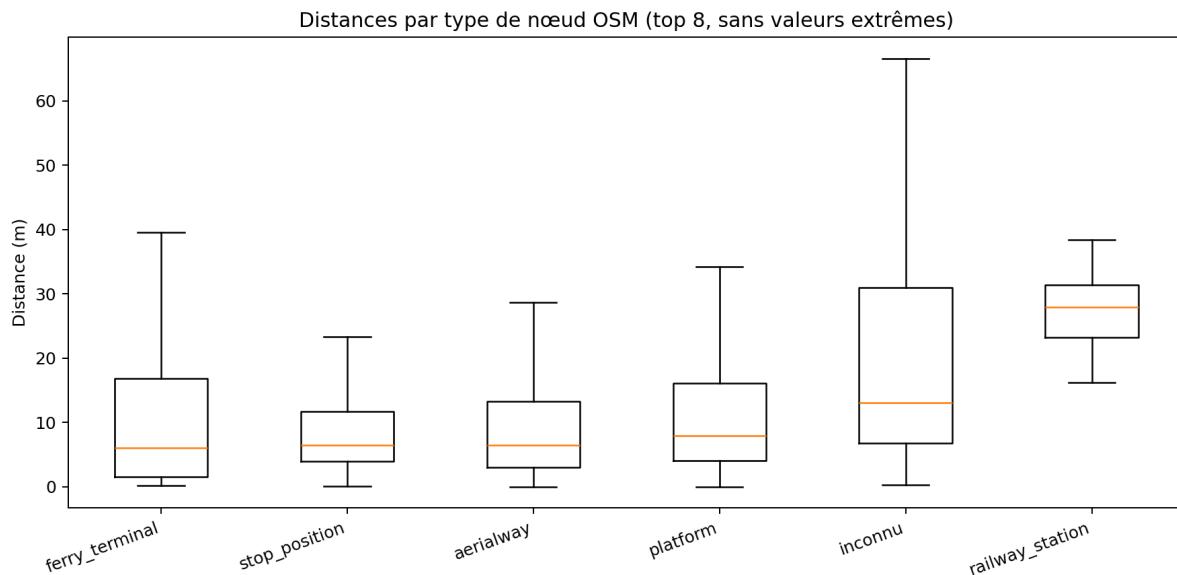


ILLUSTRATION 5.6 – Certains types (`stop_position`, `platform`) resserrent nettement les distances.

5.7. RÉFLEXIONS ET PISTES D’AMÉLIORATION

- **Homonymies et « exacts » éloignés** — compléter la règle par des garde-fous spatiaux simples (ex. : rejet si $> 1 \text{ km}$) et par une disambiguation toponymique (commune, canton).
- **Propagation contrôlée des doublons** — les cas « `duplicate_propagation` » doivent être bornés spatialement et validés à l’échelle du couloir de ligne.
- **Voisinage des non-matchés** — les 20–35% proches d’un match peuvent bénéficier d’une auto-suggestion (« propose un rattachement ») avec validation humaine.
- **Boucle de qualité continue** — mettre en place des seuils cibles ($\leq 10 \text{ m}$ pour 80% des cas, $\leq 50 \text{ m}$ pour 95%) et un tableau de bord hebdomadaire.
- **Spécificités opérateurs/terrain** — affiner les heuristiques selon les familles d’opérateurs (urbain, montagne, lacustre) et les types de nœuds OSM les plus fiables.

En résumé, l’alignement est globalement bon (médianes $\leq 10 \text{ m}$ pour les méthodes robustes), avec une traîne que l’on sait désormais localiser et traiter. Les scripts livrés rendent ces constats reproductibles et extensibles.

CHAPITRE 6 : DÉTECTION DES PROBLÈMES

Ce chapitre propose un tour d'horizon concis, visuel et orienté code de la manière dont nous détectons les problèmes de qualité des données après l'appariement des arrêts ATLAS avec les nœuds OSM. Nous présentons les types de problèmes, expliquons pourquoi et comment nous les priorisons, montrons de brefs extraits des règles, et concluons par une réflexion et des pistes d'amélioration concrètes. Attendez-vous à de petits extraits de code, des graphes compacts et un modèle mental simple pour trier des milliers d'anomalies.

6.1. CE QUE NOUS DÉTECTONS (TYPES DE PROBLÈMES)

Notre pipeline signale les problèmes au niveau consolidé des `stops` et les enregistre dans une table normalisée `problems`. Quatre familles sont actuellement prises en charge :

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

Chaque problème détecté possède un *type*, une *solution* optionnelle (pouvant être renseignée plus tard via des « solutions persistantes »), et une **priorité** numérique (P1 la plus haute, P3 la plus basse) afin de concentrer l'effort.

6.2. POURQUOI PRIORISER

Les problèmes sont nombreux et n'ont pas tous la même urgence. Un écart de 120 m dans un pôle d'échanges très fréquenté peut être bien plus impactant qu'une légère différence de chaîne de caractères d'opérateur. Les règles de priorité capturent cette intuition pour que l'interface web fasse ressortir d'abord ce qui compte.

6.3. COMMENT LES PRIORITÉS SONT ATTRIBUÉES

Nous calculons les priorités par famille de problèmes à l'aide de règles simples et interprétables. Ci-dessous, des extraits compacts qui reflètent la logique de production.

a. Priorités pour la distance

Intuition : de grands écarts spatiaux sont suspects ; nous sommes plus stricts pour les entrées opérées par les CFF (rail).

Priorisation distance — compute_distance_priority

```

1 # P1 : >80 m pour non-CFF
2 # P2 : 25--80 m pour non-CFF
3 # P3 : >25 m pour CFF OU 15--25 m pour tout opérateur
4 def compute_distance_priority(record):
5     d = float(record["distance_m"])
6     is_sbb = (record["csv_business_org_abbr"].upper() == "SBB")
7     if d > 80 and not is_sbb:
8         return 1
9     if 25 < d <= 80 and not is_sbb:
10        return 2
11    if d > 25 and is_sbb:
12        return 3
13    if 15 < d <= 25:
14        return 3
15    return None # aucun problème de distance

```

Ceci reflète le code dans `matching_process/problem_detection.py` et s'applique quand une paire est étiquetée `matched`. Les résolutions de problèmes peuvent être rendues **persistentes** afin d'être réappliquées lors des prochains imports (voir § 6.6).

b. Priorités pour les non-appariements

Intuition : être isolé sans pendant proche est plus sévère ; la distance absolue et la disponibilité UIC/plateformes aident à classer l'urgence.

Règles — non-appariements ATLAS/OSM

```

1 # Entrée ATLAS uniquement
2 if nearest_osm_distance is None or nearest_osm_distance > 80:
3     P1
4 elif nearest_osm_distance > 50:
5     P2
6 elif uic_platform_counts_mismatch:
7     P2
8 else:
9     P3
10
11 # Noeud OSM uniquement
12 if atlas_count_for_same_uic == 0:
13     P1
14 elif nearest_atlas_distance is None or nearest_atlas_distance > 50:
15     P2

```

```

16 elif uic_platform_counts_mismatch:
17     P2
18 else:
19     P3

```

Les distances au plus proche voisin sont calculées avec un KD-tree sur des coordonnées de sphère unitaire (rapide et numériquement stable). Le rayon d’isolement utilisé ailleurs est de 50 m :

Calcul voisinage — KD-tree et rayon d’isolement

```

1 ISOLATION_CHECK_RADIUS_M = 50
2 radius_rad = 2 * sin((radius_m / 6371000.0) / 2)
3 indices = kdtree.query_ball_point(unit_xyz(point), radius_rad)
4 is_isolated = (len(indices) == 0)

```

c. Priorités pour les attributs

Intuition : certains champs sont des identifiants (UIC, noms officiels) et pèsent davantage que des champs d’affichage ou opérationnels.

Priorisation attributs — compute_attributes_priority

```

1 # P1 : UIC different OU nom officiel different
2 # P2 : \texttt{local\_ref} different
3 # P3 : opérateur different
4 def compute_attributes_priority(record):
5     if record["number"] != record["osm_uic_ref"]:
6         return 1
7     if lower(record["csv_designation_official"]) != lower(record["osm_uic_name"]):
8         return 1
9     if lower(record["csv_designation"]) != lower(record["osm_local_ref"]):
10        return 2
11    if lower(record["csv_business_org_abbr"]) != lower(record["osm_operator"]):
12        return 3
13    return None

```

6.4. CE QUE CELA DONNE DANS L’APPLICATION WEB

L’interface propose des filtres simples pour permettre une triage 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.

Nous insérerons ici des captures d'écran de l'interface (filtres et listes de problèmes). Pour référence, les ressources existantes sont montrées ci-dessous et pourront être mises à jour ultérieurement :

6.5. CHIFFRES DE LA DERNIÈRE EXÉCUTION

Voici les compteurs de haut niveau affichés après l'import (pour traçabilité) :

TABLEAU 6.1 – Compteurs de haut niveau 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

Ventilations détaillées calculées automatiquement :

- **Problèmes de distance par palier** : P1=**1 171**, P2=**4 179**, P3=**4 774**; distance médiane=**30.47** m, p90=**96.64** m.
- **Non-appariements (ATLAS vs OSM)** : ATLAS uniquement=**8 416**, OSM uniquement=**21 980**; répartition P1/P2/P3=**8 192/17 647/4 557**.
- **Problèmes d'attributs par priorité** : P1=**7 045**, P2=**197**, P3=**7 654**.
- **Opérateurs les plus concernés par la distance** (top-5) : PAG=**3 520**, SBB=**1 169**, VBZ=**572**, TPF Auto=**369**, TL=**227**.
- **Arrêts avec le plus de problèmes** (top-5) : Altenrhein, Dorf=**8**, Altenrhein, Schulhaus=**8**, Sevelen, Bahnhof=**7**, Balgach, Optik=**6**, Schlosswil, Lochi=**6**.
- **Part des arrêts avec au moins un problème** : **70.2%** (52 895 / 75 359).

Illustrations générées automatiquement lors de l'extraction des statistiques :

6.6. PERSISTANCE ET FLUX DE TRAVAIL

Deux mécanismes rendent la revue 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.

Cela est appliqué à la fin de l'import :

Réapplication des solutions persistantes

```

1 for ps in persistent_solutions:
2     matching_stops = find_stops_by(sloid=ps.sloid, osm_node_id=ps.osm_node_id)
3     for stop in matching_stops:
4         problem = find_problem(stop, type=ps.problem_type)
5         if problem:
6             problem.solution = ps.solution
7             problem.is_persistent = True

```

6.7. LE MODÈLE MENTAL D'UN RÉVISEUR

En pratique, nous suggérons ce tri :

1. Filtrer **distance / P1** et trier par distance décroissante. Corriger d'abord les erreurs flagrantes de géocodage.
2. Puis **unmatched / P1**, en ciblant les cas sans pendant par UIC ou avec des écarts très importants.
3. Traiter **attributes / P1**, notamment les divergences UIC/nom qui peuvent indiquer une mauvaise identité.
4. Épurer les seaux P2/P3 restants par passes rapides ; ajouter des notes ou marquer comme cas limites acceptés si pertinent. Lorsque pertinent, *rendre persistantes* les solutions pour éviter de les resaisir lors de futurs imports.

6.8. RÉFLEXION ET PROCHAINES ÉTAPES

Ce qui fonctionne bien. Les priorités sont simples et explicables ; les réviseurs peuvent se concentrer d'abord sur les problèmes les plus impactants. L'approche KD-tree rend les vérifications d'isolement rapides à l'échelle du pays, et les solutions persistantes éliminent les ressaïssies d'une importation à l'autre.

Pistes d'amélioration.

- *Calibrage des seuils.* Apprendre des seuils à partir de données labellisées ou les adapter au contexte (urbain vs rural, bus vs rail).
- *Vérifications spatiales enrichies.* Pour les grappes denses, compléter la distance point-à-point par des comparaisons de courts trajets ou des heuristiques de groupement de quais.
- *Robustesse opérateur et dénominations.* Étendre la normalisation (diacritiques, abréviations) et tirer parti d'alias multilingues.
- *Sémantique des doublons.* Ajouter un groupement plus malin pour distinguer les vrais doublons d'une multiplicité de quais attendue.

- *Modèle de score.* Combiner les signaux (distance, lignes, noms, UIC) dans un classement appris pour les cas ambigus ; conserver les règles en repli.
- *Meilleurs résumés.* Exporter les comptes par palier et les principaux contributeurs directement pendant l'import pour afficher des histogrammes et tableaux de bord à jour (les **XX** ci-dessus).

En résumé : cette couche de détection est déjà exploitable et scalable. Avec quelques améliorations ciblées, nous pouvons accélérer le tri, réduire le bruit et suivre les progrès avec des métriques plus riches.

Filters

Matching

Matched
 Station Matched
 Unmatched

Node Type

ATLAS
 OSM

Matching Methods

Exact
 Name-based
 Distance-based
 Manual

Distance Matching Stages

Stage 1: Group-based Proximity
 Stage 2: Exact local_ref Match
 Stage 3a: Single Candidate
 Stage 3b: Relative Distance

Top N Distances

Activate Top N Filter

Station or Stop Filter

matched x Node: atlas x exact x

ILLUSTRATION 6.1 – Filtres de l’application web : par type de problème et priorité.

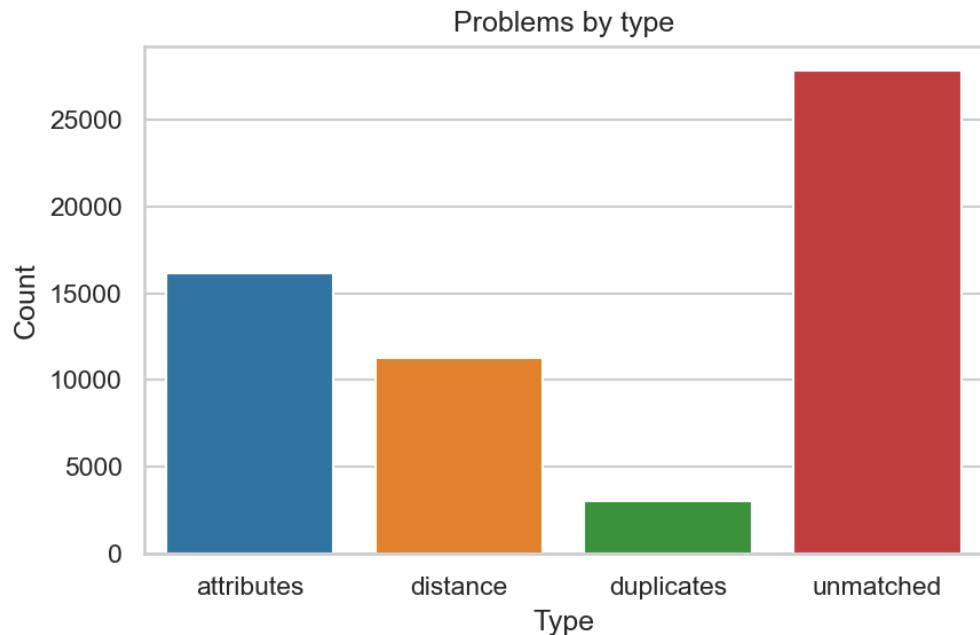


ILLUSTRATION 6.2 – Répartition des problèmes par type.

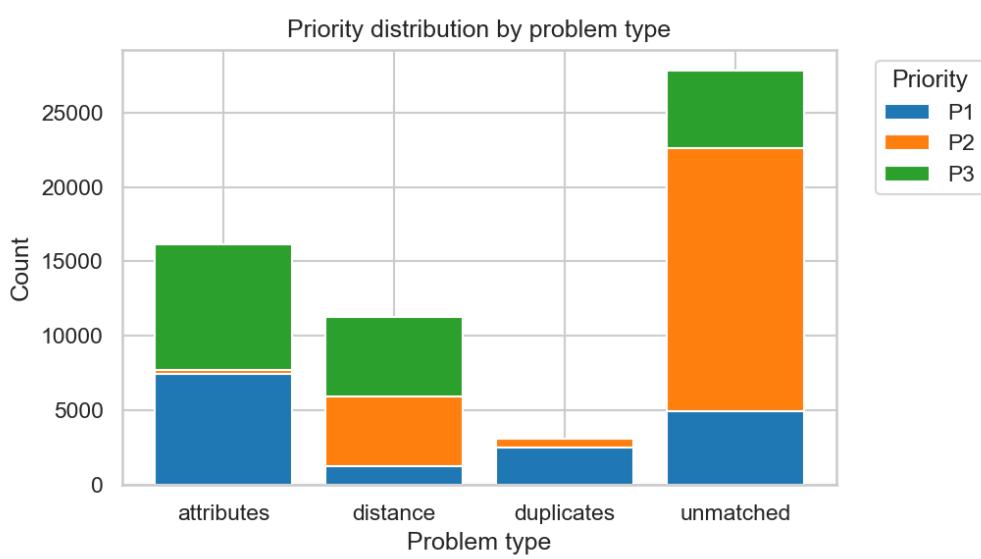


ILLUSTRATION 6.3 – Distribution des priorités (P1/P2/P3) par type de problème.

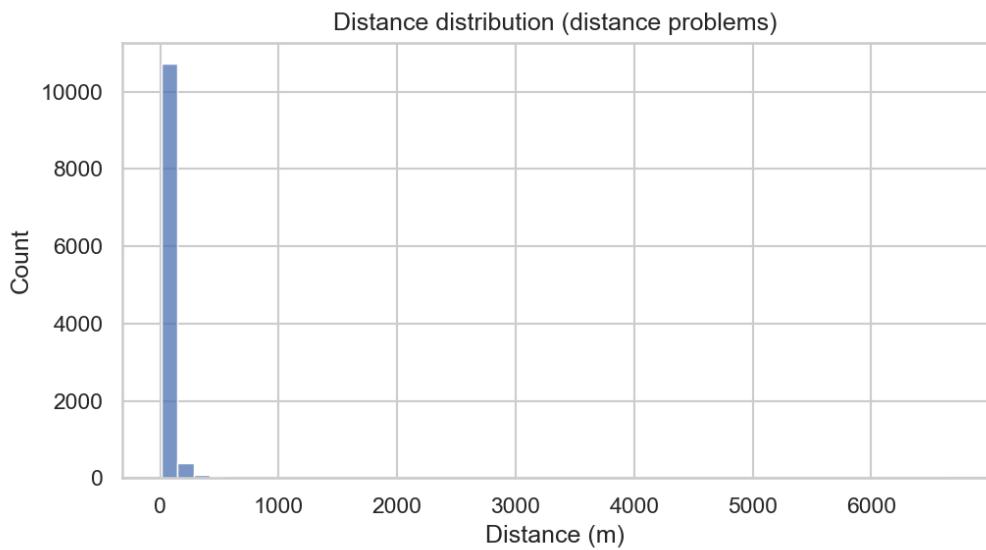


ILLUSTRATION 6.4 – Distribution des distances pour les problèmes de type distance.

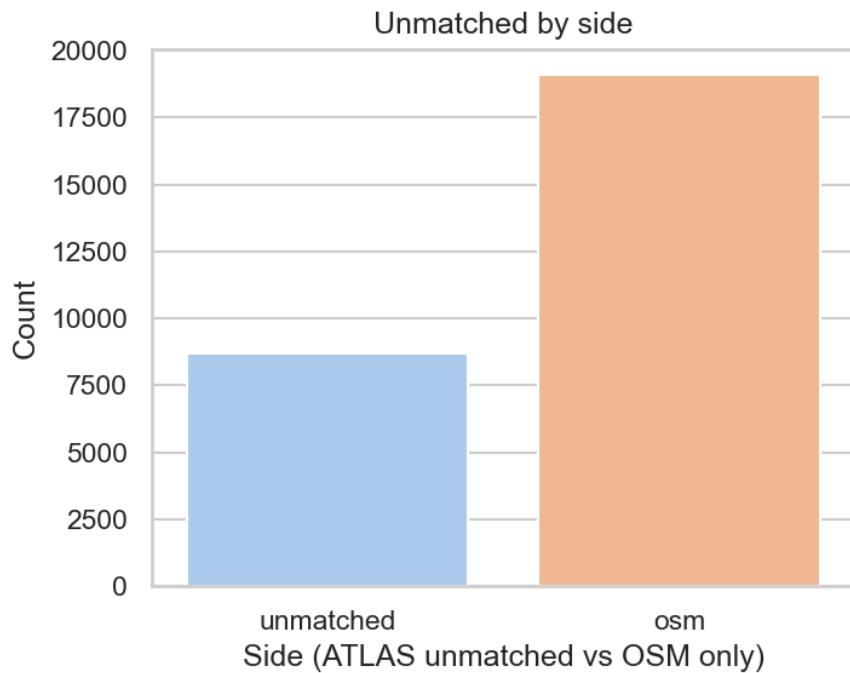


ILLUSTRATION 6.5 – Non-appariements par côté (ATLAS uniquement vs OSM uniquement).

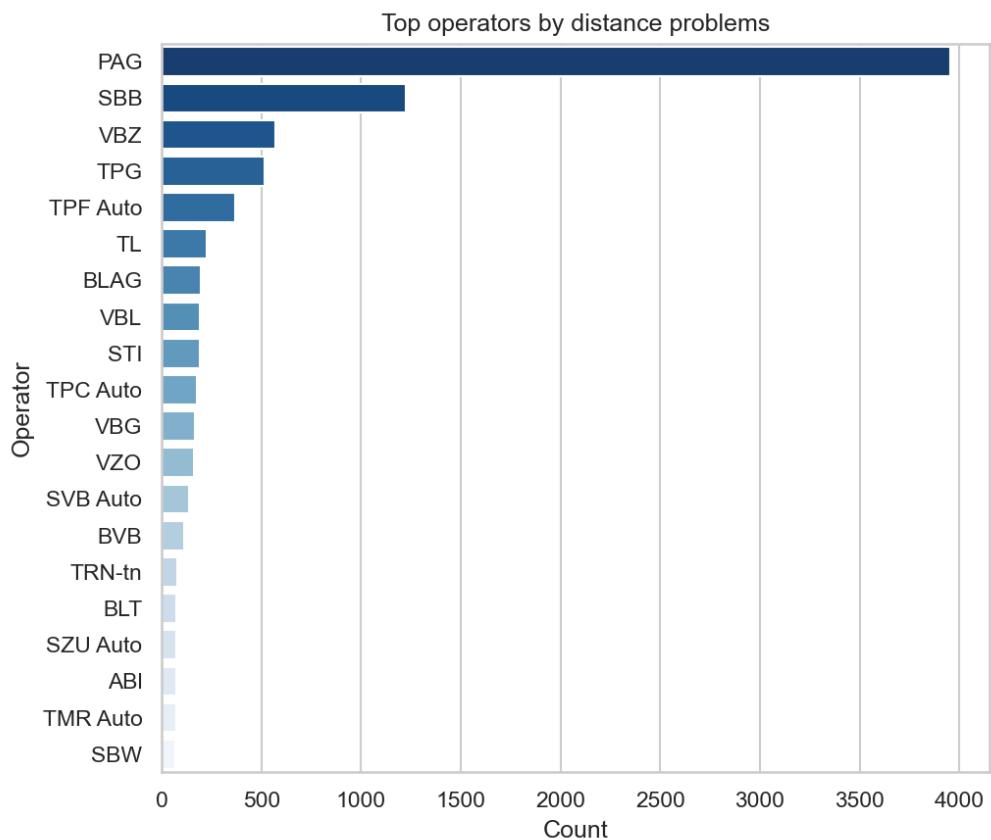


ILLUSTRATION 6.6 – Top opérateurs les plus concernés par les problèmes de distance.

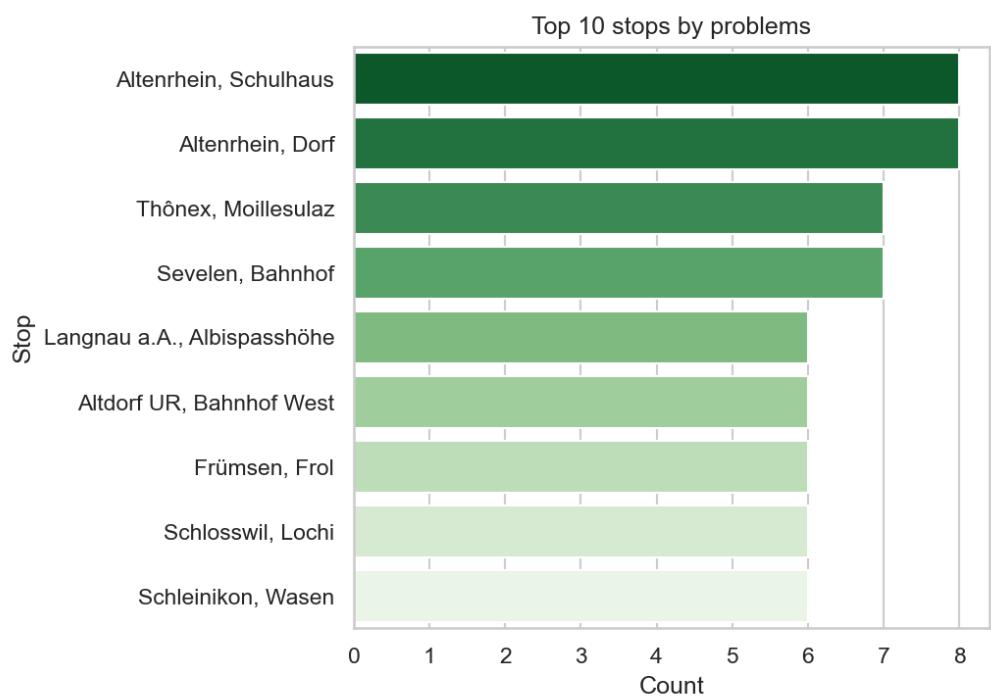


ILLUSTRATION 6.7 – Top arrêts (noms) avec le plus de problèmes.

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 provenant de plusieurs mondes : ATLAS (référentiel officiel), OpenStreetMap (OSM), GTFS et HRDF pour les lignes. 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, des schémas, et des graphiques générés directement depuis la base existante.

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

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

L'import est orchestré par le script `import_data_db.py`. Il ne se contente pas d'insérer des lignes : 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` (ATLAS isolé), `osm` (OSM isolé). Les détails riches (ex : opérateur ATLAS, tags OSM, routes) sont stockés dans des tables dédiées afin que le rendu des *popups* reste instantané et autonome.

Un extrait révélateur

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

```

Important Nous n'exécutons pas l'import dans ce chapitre (la base est déjà peuplée). Les statistiques et graphiques ci-dessous proviennent d'un script d'analyse dédié (§7.1).

7.2. SCHÉMA LOGIQUE : LES TABLES QUI COMPTENT

Le schéma applicatif est défini dans `backend/models.py` et utilise l'ORM SQLAlchemy¹⁷ 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.

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

17. *SQLAlchemy Documentation*. URL : <https://www.sqlalchemy.org/> (visité le 04/08/2025).

routes_and_directions Consolidation GTFS/HRDF/OSM par (route_id, direction_id) ou (line_name, direction_uic).

Diagramme conceptuel

Schéma relationnel des tables principales

```

1 +-----+ +-----+
2 | atlas_stops |<----->| stops |
3 | (sloid PK) | sloid | (id PK) |
4 +-----+ | sloid, osm_node_id |
5 | | atlas_* osm_* |
6 +-----+ osm_id | distance_m |
7 | osm_nodes |<----->| stop_type, ... |
8 | (osm_node_id PK) | +-----+
9 +-----+ | |
10 | | stop_id |
11 +-----+ +-----v-----+
12 | problems |<----->| stops |
13 | (stop_id FK) | | (id PK) |
14 +-----+ +-----+
15
16 +-----+
17 | persistent_data | <- Solutions/notes persistantes
18 | (sloid, osm_node_id, | entre re-imports
19 | problem_type, note_*) |
20 +-----+
21
22 +-----+
23 | routes_and_directions | <- Consolidation GTFS/HRDF/OSM
24 | (route_id, direction) | pour filtres UI
25 +-----+

```

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.

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`.

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 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.

Changement 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

PANORAMA

Ce chapitre présente l'architecture backend qui alimente l'application : structure par *blueprints* Flask,¹⁸ 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 (bases `stops_db` et `auth_db`), les extensions (SQLAlchemy, CSRF, Limiter, Talisman, Migrate) et l'enregistrement des blueprints.

Configuration Flask — `backend/app.py`

```

1 # Initialisation de l'application Flask
2 app = Flask(__name__,
3             template_folder='../../templates',
4             static_folder='../../static')
5
6 # Configuration des bases de données
7 app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URI', ...)
8 app.config['SQLALCHEMY_BINDS'] = {
9     'auth': os.getenv('AUTH_DATABASE_URI', ...)
10 }
11
12 # Initialisation des extensions
13 db.init_app(app)
14 login_manager.init_app(app)
15 csrf.init_app(app)
16 limiter.init_app(app)
17 talisman.init_app(app, content_security_policy=None)
18 migrate.init_app(app, db)
19
20 # Enregistrement des blueprints
21 app.register_blueprint(data_bp)
22 app.register_blueprint(reports_bp)
```

18. *Flask Documentation*. URL : <https://flask.palletsprojects.com/> (visité le 05/08/2025).

```

23 app.register_blueprint(search_bp)
24 app.register_blueprint(stats_bp)
25 app.register_blueprint(problems_bp)
26 app.register_blueprint(auth_bp)

```

Extensions (backend/extensions.py) :

- db = SQLAlchemy() (accès données et migrations via Alembic/Migrate)
- login_manager, csrf, limiter, talisman

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.

Sérialisation JSON — backend/serializers/stops.py

```

1 def format_stop_data(stop: Stop,
2                     problem_type: str = None,
3                     include_routes: bool = True,
4                     **kwargs):
5     """Formatage standardisé d'un Stop pour l'API JSON"""
6     return {
7         "id": stop.id,
8         "sloid": stop.sloid,
9         "stop_type": stop.stop_type,
10        "match_type": stop.match_type,
11
12        # Coordonnées (fallback OSM si pas d'ATLAS)
13        "atlas_lat": stop.atlas_lat if stop.atlas_lat is not None else stop.osm_lat
14        ,
15        "atlas_lon": stop.atlas_lon if stop.atlas_lon is not None else stop.osm_lon
16        ,
17        "osm_lat": stop.osm_lat,
18        "osm_lon": stop.osm_lon,
19
20        # Métadonnées
21        "distance_m": stop.distance_m,
22        "uic_ref": stop.uic_ref,
23        "osm_node_id": stop.osm_node_id,
24        "osm_node_type": stop.osm_node_type,
25
26        # ... (routes, notes, auteurs selon contexte)
27    }

```

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 ; filtre par fenêtre, types, opérateurs, lignes/directions.

Endpoint principal des données cartographiques

```

1 @data_bp.route('/api/data', methods=['GET'])
2 @limiter.limit("30/minute")
3 def get_data():
4     """Endpoint principal pour l'alimentation de la carte interactive"""
5
6     # 1) Parse viewport (bbox ou min/max lat/lon)
7     viewport = parse_viewport_params(request.args)
8
9     # 2) Construire les conditions (node_type, transport_types, opérateurs, routes)
10    query_conditions = QueryBuilder.build_filters(request.args)
11
12    # 3) Filtrer sur la fenêtre: index lat/lon Atlas OU OSM (SARGable)
13    viewport_query = apply_viewport_filter(query_conditions, viewport)
14
15    # 4) Offset/limit (pagination) et sérialisation légère
16    paginated_stops = apply_pagination(viewport_query, request.args)
17    serialized_stops = [format_stop_data(stop) for stop in paginated_stops]
18
19    return jsonify({
20        'stops': serialized_stops,
21        'total_count': viewport_query.count(),
22        'viewport': viewport
23    })

```

Pourquoi c'est rapide ? Le choix du schéma (Chap. 7) permet de renvoyer des objets stops autonettoyés pour le rendu, sans JOIN coûteux.

b. Recherche et matches manuels

Fichier : backend/blueprints/search.py. Gestion des recherches textuelles et des correspondances manuelles.

Endpoints de recherche — backend/blueprints/search.py

```

1 @search_bp.route('/api/search')
2 def search_stops():
3     """Recherche textuelle large (ATLAS/OSM)"""
4     # Recherche par nom, opérateur, UIC, etc.
5
6 @search_bp.route('/api/top_matches')
7 def get_top_matches():
8     """Meilleurs matches par distance et filtres"""
9     # Tri par distance croissante + filtres actifs
10
11 @search_bp.route('/api/manual_match', methods=['POST'])
12 def manual_match():
13     """Correspondance manuelle ATLAS <-> OSM, option persistance"""
14     # Logique de match manuel (voir extrait ci-dessous)

```

Extrait : logique de match manuel persistant

```

1 # Dans manual_match() -- backend/blueprints/search.py
2 atlas_stop = get_atlas_stop_by_id(atlas_stop_id)
3 osm_stop = get_osm_stop_by_id(osm_stop_id)
4
5 # Mise à jour du type de correspondance
6 atlas_stop.stop_type = 'matched'
7 atlas_stop.match_type = 'manual'
8 osm_stop.stop_type = 'matched'
9 osm_stop.match_type = 'manual'
10
11 # Option de persistance pour survivre aux ré-imports
12 if make_persistent:
13     atlas_stop.manual_is_persistent = True
14     osm_stop.manual_is_persistent = True
15
16 db.session.commit()

```

c. Statistiques : /api/global_stats

Fichier : backend/blueprints/stats.py. Statistiques agrégées, avec un cache LRU maison pour args identiques.

Cache et statistiques globales — backend/blueprints/stats.py

```

1 _STATS_CACHE = OrderedDict()
2 _STATS_CACHE_LOCK = threading.Lock()
3
4 def _build_stats_cache_key(args) -> tuple:
5     """Clé canonique basée sur les filtres triés"""
6     return tuple(sorted(args.items()))
7
8 @stats_bp.route('/api/global_stats')
9 @limiter.limit("30/minute")
10 def get_global_stats():
11     """Statistiques globales avec mise en cache LRU"""
12
13     # 1) Vérification cache -> LRU
14     cache_key = _build_stats_cache_key(request.args)
15     if cache_key in _STATS_CACHE:
16         return jsonify(_STATS_CACHE[cache_key])
17
18     # 2) Appliquer filtres partagés via QueryBuilder
19     base_query = QueryBuilder.apply_common_filters(
20         db.session.query(Stop), request.args
21     )
22
23     # 3) Compter distincts par type (ATLAS, OSM, matched pairs)
24     stats = {
25         'total_stops': base_query.count(),
26         'matched_pairs': base_query.filter(Stop.stop_type == 'matched').count(),
27         'atlas_only': base_query.filter(Stop.stop_type == 'unmatched').count(),
28         'osm_only': base_query.filter(Stop.stop_type == 'osm').count(),
29     }
30
31     # 4) Mise en cache et retour
32     with _STATS_CACHE_LOCK:
33         _STATS_CACHE[cache_key] = stats
34         if len(_STATS_CACHE) > 50: # LRU éviction
35             _STATS_CACHE.popitem(last=False)
36
37     return jsonify(stats)

```

d. Rapports : /api/generate_report

Fichier : backend/blueprints/reports.py. Génération PDF/CSV (pdfkit) sur des vues utiles (exact, noms, doublons ATLAS). La logique réutilise optimize_query_for_-

endpoint pour limiter les colonnes.

Génération de rapports — backend/blueprints/reports.py

```

1 @reports_bp.route('/api/generate_report')
2 @login_required # Auth requis désormais pour limiter l'abus
3 @limiter.limit("20/day")
4 def generate_report():
5     """Génération de rapports PDF/CSV personnalisés"""
6
7     report_format = request.args.get('format', 'pdf')
8     report_type = request.args.get('type', 'overview')
9
10    # Optimisation: sélection des colonnes nécessaires uniquement
11    optimized_query = optimize_query_for_endpoint(
12        base_query=get_filtered_stops_query(request.args),
13        columns_needed=['atlas_designation', 'osm_name', 'distance_m', ...]
14    )
15
16    data_for_report = optimized_query.all()
17
18    if report_format == 'csv':
19        # Génération CSV avec en-têtes adaptées
20        csv_content = generate_csv_content(data_for_report, report_type)
21        return Response(
22            csv_content,
23            mimetype='text/csv',
24            headers={'Content-Disposition': f'attachment; filename=report_{report_type}.csv'}
25        )
26    else:
27        # Génération PDF via template HTML
28        report_html = render_template(
29            'pages/report.html',
30            report_items=data_for_report,
31            report_type=report_type,
32            generated_at=datetime.now()
33        )
34
35        pdf_bytes = pdfkit.from_string(report_html, False, options={
36            'page-size': 'A4',
37            'orientation': 'Landscape',
38            'margin-top': '0.75in'
39        })
40
41        return Response(

```

```

42         pdf_bytes,
43         mimetype='application/pdf',
44         headers={'Content-Disposition': f'attachment; filename=report_{
45             report_type}.pdf'}
        )

```

e. Problèmes et persistance

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

API de gestion des problèmes — backend/blueprints/problems.py

```

1 @problems_bp.route('/api/problems')
2 def get_problems():
3     """Liste paginée des problèmes (tri par distance/priorité)"""
4     # Tri par priorité décroissante, puis distance croissante
5
6 @problems_bp.route('/api/problems/stats')
7 def get_problems_stats():
8     """Taux résolus / non résolus par type"""
9     # Agrégation par problem_type et status
10
11 @problems_bp.route('/api/save_solution', methods=['POST'])
12 @csrf.exempt # AJAX endpoint
13 def save_solution():
14     """Enregistre une solution (non persistée par défaut)"""
15     # Mise à jour Problem.solution, Problem.solved_by
16
17 @problems_bp.route('/api/make_solution_persistent', methods=['POST'])
18 @csrf.exempt
19 def make_solution_persistent():
20     """Rend une solution persistante entre ré-imports"""
21     # Insertion dans PersistentData
22
23 @problems_bp.route('/api/save_note/<note_type>', methods=['POST'])
24 @csrf.exempt
25 def save_note(note_type):
26     """Sauvegarde note ATLAS ou OSM (note_type: 'atlas'|'osm')"""
27     # Mise à jour atlas_stops.note ou osm_nodes.note
28
29 @problems_bp.route('/api/make_note_persistent/<note_type>', methods=['POST'])

```

```

30 @csrf.exempt
31 def make_note_persistent(note_type):
32     """Rend une note persistante entre ré-imports"""
33     # Insertion dans PersistentData avec note_type

```

8.3. REQUÊTAGE PARTAGÉ ET OPTIMISATION

QueryBuilder et FilterBuilder

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

Filtres partagés — backend/query_builder.py

```

1 class QueryBuilder:
2     """Constructeur de requêtes avec filtres standardisés"""
3
4     transport_mappings = {
5         'station': Stop.osm_node_details.has(
6             and_(OsmNode.osm_public_transport == 'station',
7                  OsmNode.osm_railway.in_(['station', 'halt'])))
8         ),
9         'platform': Stop.osm_node_details.has(
10            OsmNode.osm_public_transport == 'platform'
11        ),
12         'stop_position': Stop.osm_node_details.has(
13            OsmNode.osm_public_transport == 'stop_position'
14        ),
15         # ... autres types
16     }
17
18     @classmethod
19     def apply_common_filters(cls, query, filters_dict):
20         """Application des filtres standardisés"""
21         conditions = []
22
23         # Filtres par type de transport
24         if 'transport_types' in filters_dict:
25             transport_conditions = [
26                 cls.transport_mappings[t]
27                 for t in filters_dict['transport_types']
28                 if t in cls.transport_mappings

```

```

29         ]
30     if transport_conditions:
31         conditions.append(or_(*transport_conditions))
32
33     # Filtres par opérateur ATLAS
34     if 'operators' in filters_dict:
35         conditions.append(
36             Stop.atlas_stop_details.has(
37                 AtlasStop.operator.in_(filters_dict['operators']))
38         )
39     )
40
41     # Application finale
42     if conditions:
43         query = query.filter(and_(*conditions))
44
45     return query

```

Services de routes

Fichier : backend/services/routes.py. Fournit `get_stops_for_route(route_id, direction)` en SQL brut pour aller vite dans `routes_and_directions`. Supporte un *fallback* de normalisation d'ID (ex : `-j24 → -jXX`).

Requête optimisée pour les arrêts d'une ligne

```

1 -- Service get_stops_for_route() -- backend/services/routes.py
2 SELECT
3     osm_nodes_json,
4     atlas_sloids_json,
5     osm_route_id,
6     atlas_route_id,
7     atlas_line_name
8 FROM routes_and_directions
9 WHERE
10    osm_route_id LIKE :route_id
11    OR atlas_route_id LIKE :route_id
12    OR atlas_line_name LIKE :route_id
13    OR atlas_line_name LIKE :normalized_route_id -- fallback normalisé
14 LIMIT 1;

```

Logique de normalisation et fallback

```

1 def get_stops_for_route(route_id: str, direction: str = None) -> dict:
2     """Récupération optimisée des arrêts pour une route donnée"""
3
4     # Tentative directe
5     result = db.session.execute(text(ROUTE_QUERY), {
6         'route_id': f'%{route_id}%',
7         'normalized_route_id': f'%{normalize_route_id(route_id)}%'
8     }).first()
9
10    if result:
11        return {
12            'osm_nodes': json.loads(result.osm_nodes_json or '[]'),
13            'atlas_sloids': json.loads(result.atlas_sloids_json or '[]'),
14            'source': 'direct_match'
15        }
16
17    # Fallback: normalisation plus agressive
18    normalized_id = normalize_route_id(route_id, aggressive=True)
19    if normalized_id != route_id:
20        return get_stops_for_route(normalized_id, direction)
21
22    return {'osm_nodes': [], 'atlas_sloids': [], 'source': 'no_match'}
23
24 def normalize_route_id(route_id: str, aggressive: bool = False) -> str:
25     """Normalisation des IDs de route (ex: -j24 -> -jXX)"""
26     if aggressive:
27         # Remplace les suffixes journaliers: -j24 -> -jXX
28         return re.sub(r'-j\d+$', '-jXX', route_id)
29     else:
30         # Normalisation simple (espaces, casse)
31         return route_id.strip().upper()

```

8.4. DIAGRAMMES DE FLUX

Flux d'une requête cartographique

Pipeline de traitement — /api/data

```

1 Client
2   | GET /api/data?bbox=...&operators=...
3   v
4 Parse params (viewport, filters)
5   |
6   v
7 Build conditions (QueryBuilder)
8   | transport_types, operators, routes
9   v
10 Viewport filter (indexes lat/lon)
11  | SARGable query sur atlas_* ET osm_*
12  v
13 Pagination (offset/limit)
14  |
15  v
16 Serialisation JSON (format_stop_data)
17  |
18  v
19 Response -> Map rendering

```

Cycle de persistance des solutions

Workflow de persistance

```

1 +-----+      save_solution      +-----+
2 | UI (Problems)    | -----> | Problem.solution | 
3 |                   |           | (temporaire)  |
4 +-----+           +-----+
5           |           |
6           | make_persistent   |
7           v           v
8 +-----+      persist_data      +-----+
9 | PersistentData    | <-----| Problem record   |
10 | (survit import) |           | + is_persistent |
11 +-----+           +-----+
12           |
13           | Prochain import

```

```
14      v
15 +-----+    apply_solutions
16 | Nouvel import | -----> Solution reappliquée
17 | (import_data_db) |           automatiquement
18 +-----+
```

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.

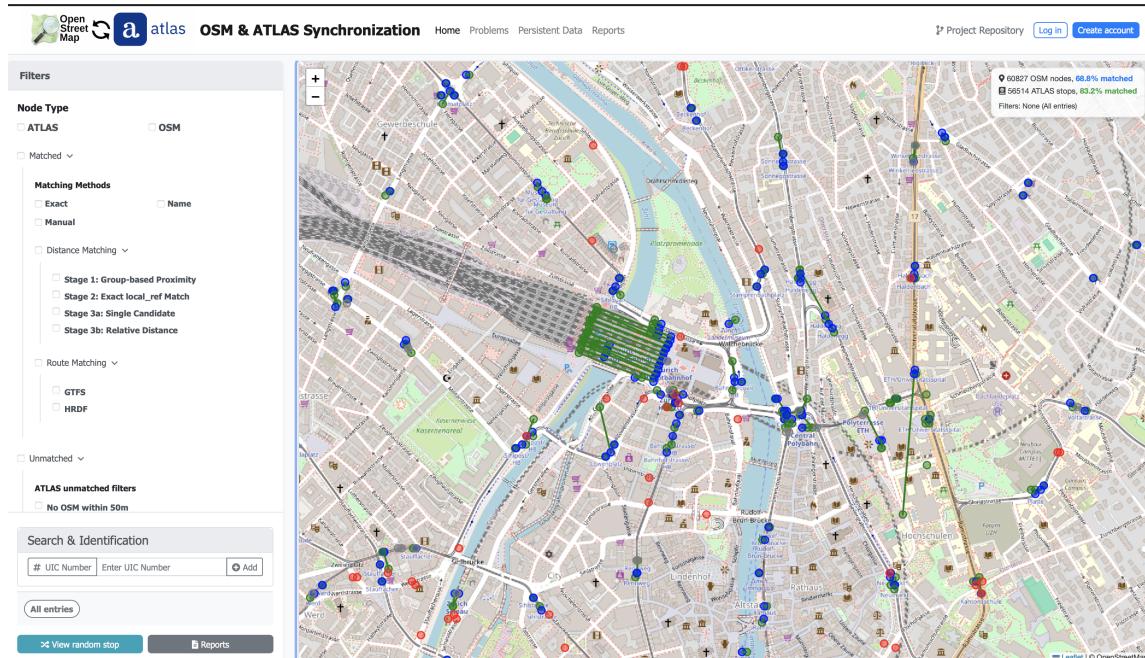


ILLUSTRATION 9.1 – Vue générale de la page d'accueil

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).

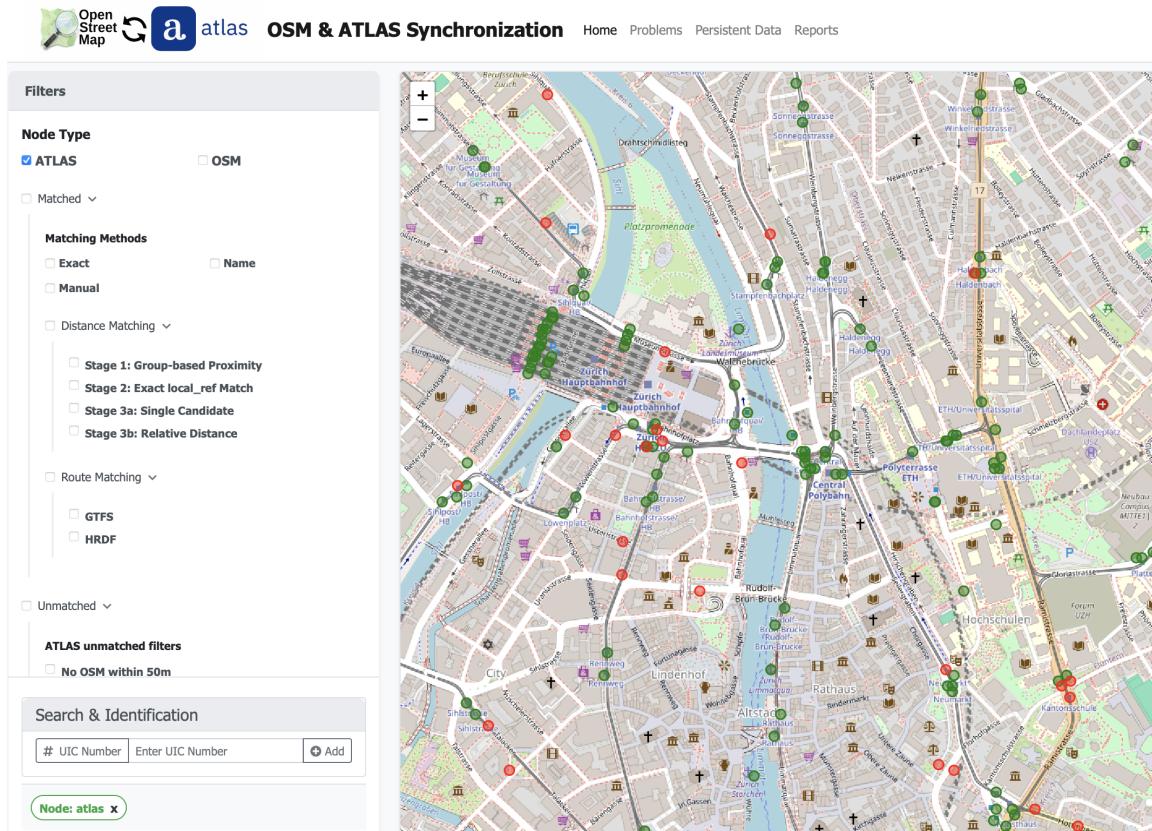


ILLUSTRATION 9.2 – Filtre ATLAS activé — ville de Zurich

Chaque page est construite en HTML Jinja,¹⁹ 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 (Leaflet²⁰) 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.
- **Milieu de zoom** — résultats plafonnés (≤ 500).

19. *Jinja Documentation*. URL : <https://jinja.palletsprojects.com/> (visité le 06/08/2025).

20. *Leaflet Documentation*. URL : <https://leafletjs.com/> (visité le 07/08/2025).

— **Fort zoom** — plafond levé : *tous* les marqueurs de la fenêtre sont renvoyés.

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

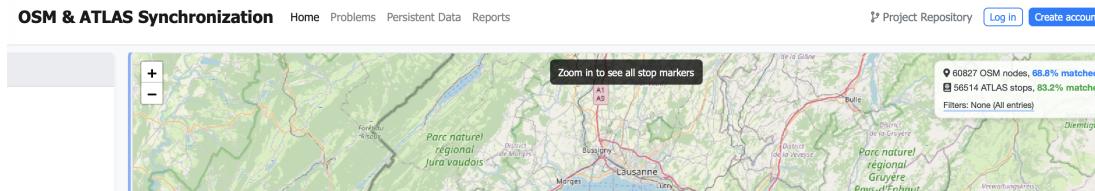


ILLUSTRATION 9.3 – Bannière politique de rendu selon le niveau de zoom

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*).
- **Méthodes d’appariement** : Exact, Name, Manual ; **Distance Matching** (stages 1, 2, 3a, 3b) ; **Route Matching** (GTFS, HRDF).

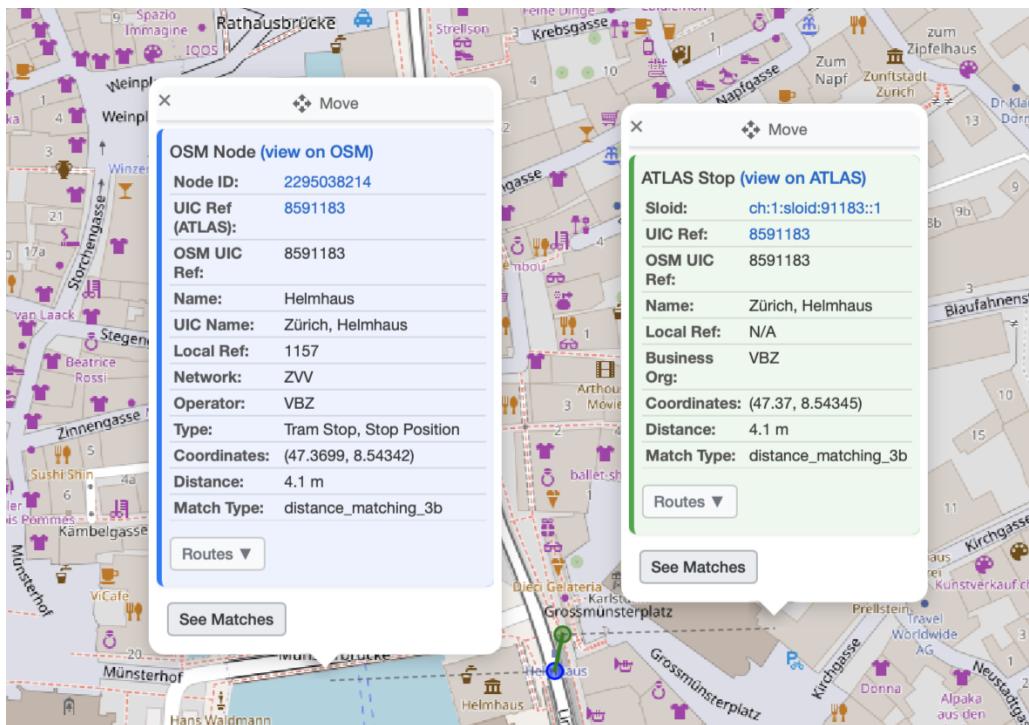


ILLUSTRATION 9.4 – Info-bulles

- **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 **persistence** des corrections, et l'édition de **notes** ATLAS/OSM. Le *toggle* Auto-Persist permet 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).

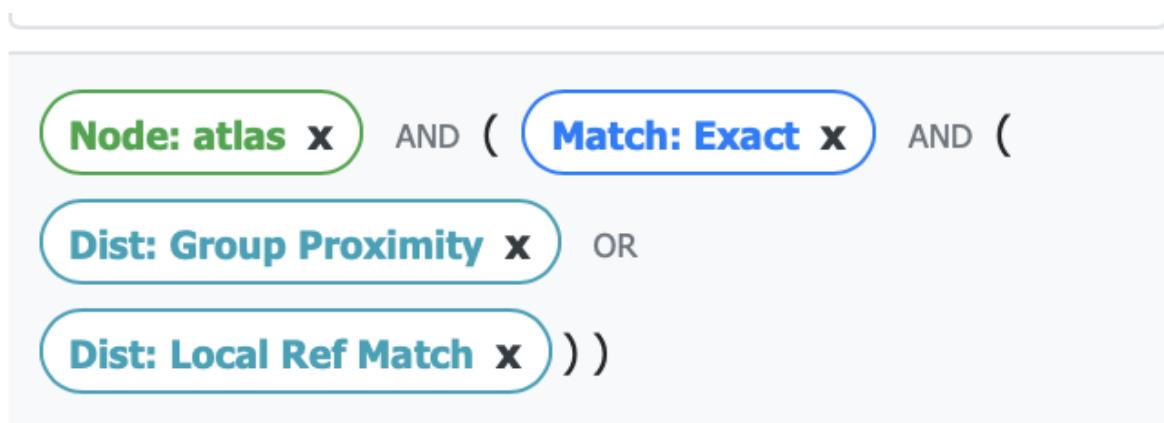


ILLUSTRATION 9.5 – Panneau de filtres avec « chips » actifs

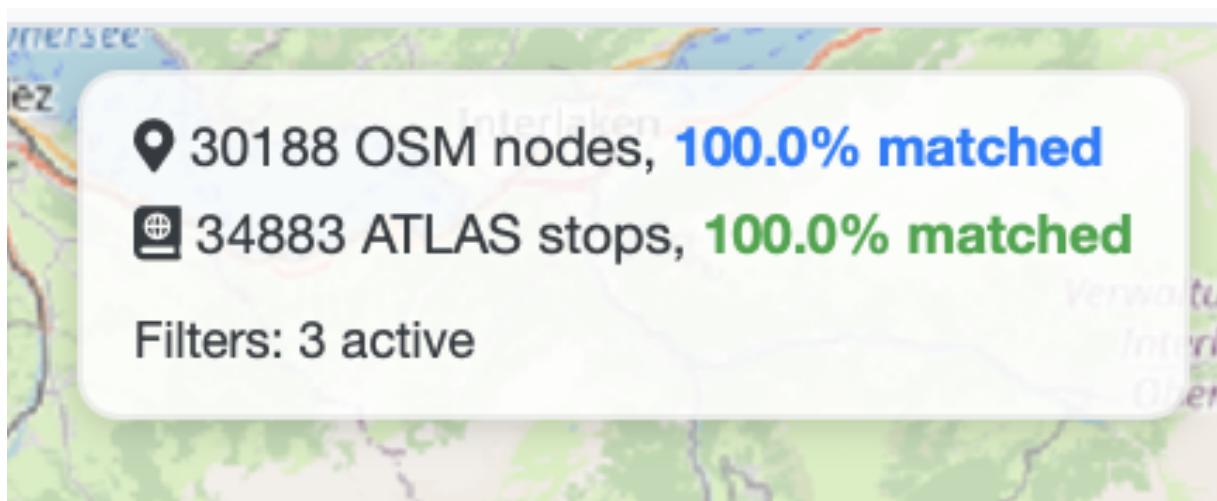


ILLUSTRATION 9.6 – Résumé d'en-tête avec statistiques globales. 3 filtres actifs

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 garantir 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 mé-

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

moire.

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

[Back to Map](#)

Filter by Problem Type:

All Problems

Priority

All **P1** P2 P3

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

Atlas Operator:

Select operators...

Active Filters: **Unmatched x** AND **Priority P1 x**

Persistence Options

Auto-Persist new solutions Auto-Persist new notes

[Manage Persistent Data](#)

Filtre de priorité et tri.

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... [Save ATLAS Note](#)

OSM Note: Add a note for this OSM entry... [Save OSM Note](#)

[Edit in OSM ID Editor](#)

Entry 1 of 7538 (1 Problem)

Attribute Comparison

Critical attribute mismatch, UIC Name.

ATLAS Entry	OSM Entry
ATLAS Stop (view on ATLAS)	OSM Node (view on OSM)
Stop ID: ch1:solid:207:4:10	Node ID: 6551129268
UIC Ref: 8500207	UIC Ref (ATLAS): 8500207
Name: Solothurn	OSM UIC Ref: 8500207
Local Ref: 10	Name: Solothurn RBS
Business Org: SBB	UIC Name: Solothurn [Gleis 9-10]
Coordinates: (47.2035, 7.54313)	Local Ref: 10
Match Type: exact	Network: Libero
No route information available	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]	<input type="checkbox"/> Use ATLAS	<input type="checkbox"/> Use OSM

Overall Status

Not a valid match [Skip](#)

Actions proposées : problèmes de distance.

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 OSM & ATLAS Synchronization web application. At the top, there are navigation links: 'Home', 'Problems', 'Persistent Data' (which is the active tab), and 'Reports'. Below the navigation, there are several buttons: 'Back to Problems', 'Map page', 'Make All Persistent' (in green), and 'Filter by Type'. A message box states: 'Persistent data is automatically applied after each data import. They help maintain permanent problem resolutions after data imports.' Another message box says: 'This data is permanently stored and will be automatically applied after future data imports.' A red error message box displays: 'Error loading persistent data: UNAUTHORIZED'. There are also tabs for 'Persistent Data' and 'Non-Persistent Data'.

ILLUSTRATION 9.10 – Données persistantes — pour utilisateurs non administrateurs

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

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 a vu, les données principales de l'application demeurent dans `stops_db`. Cette séparation réduit le rayon d'impact et garantit entre d'autres choses que la réimportation des données de transport public ne touche jamais les identifiants des utilisateurs.

a. Composants

- **Stockage des mots de passe** : Argon2id (argon2-cffi),²¹ 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)²² avec activation par QR code et codes de secours à usage unique. **Le secret TOTP est chiffré au repos** (Fernet)²³ ; 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 Turnstile²⁴ sur /auth/register et /auth/login.

21. Argon2 documentation. URL : <https://argon2-cffi.readthedocs.io/en/stable/> (visité le 17/06/2025).

22. RFC 6238 : TOTP : Time-Based One-Time Password Algorithm. URL : <https://www.rfc-editor.org/rfc/rfc6238> (visité le 15/06/2025).

23. Fernet (symmetric encryption). URL : <https://cryptography.io/en/latest/fernet/> (visité le 18/06/2025).

24. Cloudflare Turnstile. URL : <https://www.cloudflare.com/products/turnstile/> (visité le

- **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)

```

```
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.

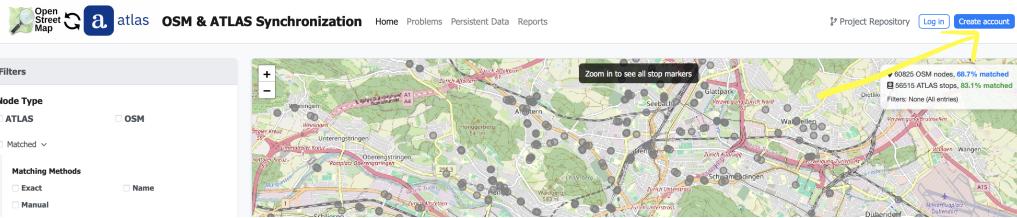


ILLUSTRATION 10.1 – Bouton « Créer un compte » visible dans l'interface.

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)
```

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.

Placeholder visuel pour la capture de la vérification d'email.
Lien signé, expiration 48 h, renouvelable.

ILLUSTRATION 10.4 – Processus de vérification d'email (espace réservé).

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).²⁵ Les emails de vérification sont rendus côté serveur puis envoyés via Amazon Simple Email Service (SES) grâce au SDK boto3. La fonction 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.

25. Amazon Simple Email Service (SES). URL : <https://aws.amazon.com/ses/> (visité le 11/08/2025).

<input type="checkbox"/> is_admin	<input type="checkbox"/> is_email_verified	<input type="checkbox"/> email_verified_at	<input type="checkbox"/> last_verification_sent_at	<input type="checkbox"/> is_totp_enabled
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.

OSM & ATLAS Synchronization Home Problems Persistent Data Reports Project Repository Log in Create account

Login

Email
bonjourguillem+2@gmail.com

Password

Remember me

Login

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

OSM & ATLAS Synchronization Home Problems Persistent Data Reports Project Repository bonjourguillem+2@gmail.com Unverified Enable 2FA Logout

Your email is not verified yet. You can continue; we sent a verification email.

Logged in successfully.

Map showing 60825 OSM nodes, 88.7% matched and 56515 ATLAS stops, 83.1% matched. Filters: None (All entries).

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.

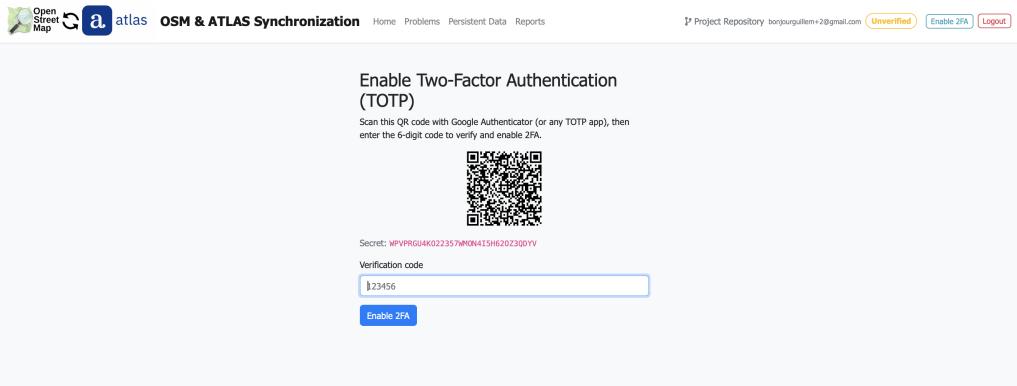


ILLUSTRATION 10.9 – Activation 2FA avec QR *otpauth://* et secret Base32. QR et secret encodent la même information.

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)
```

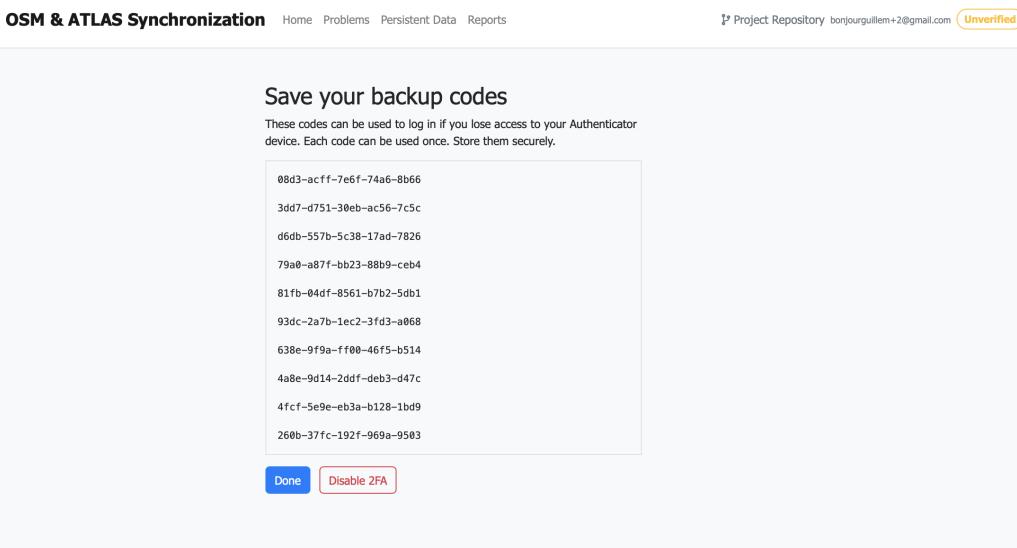


ILLUSTRATION 10.10 – Codes de secours : affichés une seule fois à l’activation. Stockage côté serveur : hachés Argon2 dans un JSON.

The screenshot shows a user interface for a two-factor authentication (2FA) step. At the top, it says "Enter 2FA code". Below that is a "Token" label with an input field containing the value "123 456". Underneath the input field is a note: "You can also enter a backup code if you lost device access." At the bottom is a blue "Verify" button.

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

totp_secret	backup_codes_json	created_at	updated_at
25CX4EDGL6IKNR4JHX6S0MD0R7BBNUAU	["\$argon2id\$v=19\$m=65536,t=3,p=4\$hM0HhJTXxSD/m1bQCU..."	2025-08-16 10:24:13	2025-08-16 10:31:54
IKDNNN23CH27TG3BGD5544K6KJFQH2BJB	["\$argon2id\$v=19\$m=65536,t=3,p=4\$8Rasws14ZepM39KoXP..."	2025-08-16 10:32:36	2025-08-16 10:33:54
WPVPR6U4K022357WNON4I5H620Z3QDV	["\$argon2id\$v=19\$m=65536,t=3,p=4\$oUvg9N1S7LjEImYbcL..."	2025-08-16 18:01:28	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 CAPTCHA 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 comprendre comment et pourquoi les contrôles bloquent les attaques.

11.1. PÉRIMÈTRE ET SURFACE D'ATTAQUE

Pour rendre la lecture concrète, on part d'une *carte des routes* – celles que l'utilisateur peut toucher. 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.

Types d'attaques considérés : **boufrage d'identifiants, force brute, contournement 2FA, énumération d'email, CSRF, abus de liens signés et déni de service (DoS)**.

11.2. CONTRÔLES EN PLACE (APERÇU TECHNIQUE)

Mots de passe et stockage

Les mots de passe sont hachés avec Argon2id²⁶ 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)
```

26. Argon2 documentation, cf. note 21.

Sessions et cookies

Cookies `HttpOnly`, `SameSite=Lax` et `Secure` activable par variable d'environnement (produit : 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 — explication claire

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 (ou un autre mécanisme équivalent) pour être sûr.

À retenir. `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 Turnstile²⁷ protège inscription et 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'))
```

27. *Cloudflare Turnstile*, cf. note 24.

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. Pour la connexion, la clé de limite combine **IP + email tenté** afin d'empêcher un attaquant de détourner la limite IP seule.

Caveat proxys. 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)²⁸ via pyotp. Le secret TOTP est chiffré au repos (Fernet)²⁹ 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;
```

28. RFC 6238 : TOTP : Time-Based One-Time Password Algorithm, cf. note 22.

29. Fernet (symmetric encryption), cf. note 23.

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.

Placeholder : *Courbe du temps d’attente cumulé* vs nombre d’essais échoués.

ILLUSTRATION 11.1 – Verrouillage progressif : coût croissant pour l’attaquant.

Placeholder capture d’écran : message UI « Mot de passe incorrect » suivi d’un « Compte temporairement verrouillé ».

ILLUSTRATION 11.2 – Échec de mot de passe et blocage temporaire affiché dans l’interface.

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.

Placeholder : *Diagramme de séquence « login + 2FA » avec embranchements TOTP/backup.*

ILLUSTRATION 11.3 – Garde 2FA : étape obligatoire après mot de passe.

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.

Placeholder capture d'écran : saisie d'un code 2FA **incorrect** avec message d'erreur.

ILLUSTRATION 11.4 – Tentative avec code 2FA erroné.

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. **Exception :** /auth/register indique si l'email existe déjà.

Résultat. L'énumération est évitée sur login/resend, mais possible sur register. Voir § 11.5 pour uniformiser les messages.

Placeholder capture d'écran : tentative d'inscription avec un email existant et message révélant l'existence du compte.

ILLUSTRATION 11.5 – Exemple d'énumération d'email via message d'erreur d'inscription.

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.

Placeholder : *Timeline* d'un jeton : émission, délai de 48 h, invalidation.

ILLUSTRATION 11.6 – Cycle de vie d'un jeton de vérification.

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 métriques d'audit 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 ; ajouter une clé composite (IP + email cible) et des « seaux » par utilisateur.
- **Verrouillage par compte** : robuste, mais attention au déni de service ciblé (un adversaire peut « verrouiller » le compte d'une victime). Des *captcha* et *cooldowns* nuancés aident à mitiger.
- **Remember cookie de 14 jours** : confortable ; réduire la durée ou exiger 2FA périodiquement renforce la sécurité.

11.5. AMÉLIORATIONS ET VECTEURS NON ENCORE COUVERTS

Mesures recommandées, de l'immédiat au stratégique :

- **Forcer HTTPS et cookies Secure partout** (`FORCE_HTTPS=true`, `SESSION_COOKIE_SECURE=true`).
- **Politiques CSP strictes** (actuellement désactivées) avec listes blanches minimales, ou auto-hébergement des actifs.
- **Unifier les messages d'inscription** pour éviter l'énumération d'emails (réponse neutre : « si un compte existe déjà, vous recevrez un email »). *Implémenté*. La page d'inscription et les écrans associés utilisent un message neutre.
- **Clés de limite composites** : ajouter des limites *par compte* (IP+email) et des *burst tokens* pour lisser.
- **Écriture manuelle** : exiger l'authentification forte pour tout endpoint mutateur (p. ex. `/api/manual_match`) et journaliser finement.
- **Chiffrement au repos du secret TOTP** (Fernet, clé via variable d'environnement) et **réauthentification** exigée pour `/auth/disable_2fa`. *Implémenté*. Désactivation 2FA requiert le mot de passe.
- **Rotation de session à la connexion** et vidage explicite de session pour réduire les risques de fixation.
- **Réinitialisation de mot de passe** avec jetons signés mono-usage à durée courte (non implémenté, § 10).
- **Détection d'anomalies basée sur les journaux d'audit** (IP, ASN, pays), alertes et tableaux de bord ; politiques de rétention/archivage.
- **Facteur résistant au phishing** (*WebAuthn / Passkeys*) en option, en complément du TOTP.
- **Durcissement du pipeline email** : DKIM/DMARC stricts, liens à usage unique invalidés à la première visite.

Priorité court terme. HTTPS forcé, CSP durcie, réauthentification pour désactiver la 2FA, chiffrement du secret TOTP, messages d'inscription non « révélateurs ».

11.6. 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 ».

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`. Ce chapitre couvrira également le lancement du pipeline complet, l'utilisation d'un mode de développement léger, et les techniques de diagnostic et de personnalisation.

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 conteneurisation.³⁰ 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. 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.

12.2. POURQUOI DOCKERISER ?

Trois raisons concrètes :

- **Reproductibilité** : même version de Python, mêmes dépendances système (`wkhtmltopdf`,³¹ 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.

12.3. 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'environnement,³² `depends_on` avec `healthcheck`.

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

30. *Docker Documentation*. URL : <https://docs.docker.com/> (visité le 12/08/2025).

31. *wkhtmltopdf*. URL : <https://wkhtmltopdf.org/> (visité le 13/08/2025).

32. *Environment variables with Compose*. URL : <https://docs.docker.com/compose/environment-variables/envvars/> (visité le 14/08/2025).

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

12.4. APERÇU

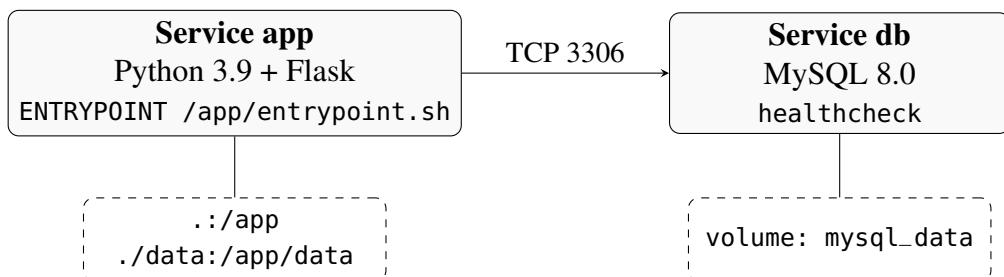


ILLUSTRATION 12.1 – Deux conteneurs reliés sur le réseau compose ; volumes pour le code (monté) et les données MySQL (persistées).

12.5. L'ORCHESTRATION : `docker-compose.yml`

Le fichier compose³³ 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
  
```

L'application dépend explicitement du statut *healthy* de MySQL³⁴ et monte deux volumes (code + données) :

33. *Docker Compose*. URL : <https://docs.docker.com/compose/> (visité le 15/08/2025).

34. *MySQL 8.0 Documentation*. URL : <https://dev.mysql.com/doc/> (visité le 16/08/2025).

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: ${AUTH_DATABASE_URI:-mysql+pymysql://stops_user:1234@db/
19   auth_db}
20   AUTO_MIGRATE: "true"
21   MATCH_ONLY: "${MATCH_ONLY:-false}"
22   SECRET_KEY: ${SECRET_KEY:-dev-insecure}

```

En développement, nous activons un service plus léger qui évite les téléchargements et prétrainements 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"

```

12.6. L'IMAGE : dockerfile

L'image part d'un Python 3.9 minimal `slim-bookworm` (compatibilité `wkhtmltopdf`). Nous installons trois dépendances `apt : wkhtmltopdf, default-mysql-client` (pour `mysqladmin`)

et dos2unix. 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"]

```

12.7. 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 Flask.³⁵ Il peut également créer un utilisateur dédié pour auth_db si des variables d'environnement sont

³⁵. *Flask Documentation*, cf. note 18.

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}" \\
11        -e "CREATE DATABASE IF NOT EXISTS auth_db CHARACTER SET utf8mb4 COLLATE
12          utf8mb4_unicode_ci; GRANT ALL PRIVILEGES ON auth_db.* TO 'stops_user'@'%';
13          FLUSH PRIVILEGES;" || true
14    # Crée un utilisateur dédié auth si variables fournies et révoque stops_user sur
15    # auth_db
16    if [ -n "$AUTH_DB_USER" ] && [ -n "$AUTH_DB_PASSWORD" ]; then
17        mysql -h db -uroot -p"${MYSQL_ROOT_PASSWORD}" \\
18            -e "CREATE USER IF NOT EXISTS '${AUTH_DB_USER}'@'%' IDENTIFIED BY '${
19              AUTH_DB_PASSWORD}'; GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, INDEX
20              ON auth_db.* TO '${AUTH_DB_USER}'@'%'; REVOKE ALL PRIVILEGES ON auth_db.* FROM
21              'stops_user'@'%'; FLUSH PRIVILEGES;" || true
22    fi
23    fi
24
25 if [ "${AUTO_MIGRATE:-false}" = "true" ]; then
26     [ ! -d "migrations" ] && flask db init || true
27     flask db migrate -m "Auto migration" || true
28 fi
29 flask db upgrade || true

```

Le démarrage exécute également `create_auth_tables.py` pour s'assurer que le schéma `auth_db` inclut **users** et **auth_events** (journalisation d'audit).

12.8. 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
```

12.9. CAPTURES D'ÉCRAN

```
-----(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.

The screenshot shows the Docker Desktop interface. At the top, there's a summary of CPU and memory usage: "Container CPU usage 101.14% / 800% (8 CPUs available)" and "Container memory usage 2.71GB / 3.73GB". Below this is a search bar and a filter option "Only show running containers". A table lists four containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
☰	bachelor-project	-	-	-	101.14%	59 minutes ago	⋮ ⌂
☰	bachelor_project_app_dev	6f6821345278	bachelor-project-app-dev	5001:5001	0%	11 hours ago	▷ ⋮ ⌂
☰	bachelor_project_db	3bf61d92c440	mysql:8.0	3306:3306 ⌂	0.75%	59 minutes ago	⋮ ⌂
☰	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.

Placeholder pour une image supplémentaire à insérer (par ex. vue de l'application après déploiement).

ILLUSTRATION 12.4 – [À remplacer] Capture d'écran complémentaire.

12.10. PERSONNALISER PAR 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
<code>DATABASE_URI</code>	Connexion MySQL (stops)	<code>mysql+pymysql://stops_-user:1234@db/stops_db</code>
<code>AUTH_DATABASE_URI</code>	Base d'authentification	<code>mysql+pymysql://.../auth_db</code>
<code>AUTH_DB_USER,</code> <code>AUTH_DB_PASSWORD</code>	Compte dédié auth (optionnel)	<i>vide par défaut</i>
<code>AUTO_MIGRATE</code>	Init/migrate auto Alembic ³⁶	<code>true</code> (dev)
<code>SKIP_DATA_IMPORT</code>	Sauter pipeline de données	<code>false</code> (app), <code>true</code> (app-dev)
<code>MATCH_ONLY</code>	N'exécuter que matching + import	<code>false</code>
<code>FLASK_ENV,</code> <code>FLASK_DEBUG</code>	Mode Flask	<code>development, 1</code>

12.11. 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 `.dockerignore` exclut documentation et gros dossiers de données (tout en gardant quelques fichiers essentiels pour MATCH_ONLY). Cela accélère docker build et évite des images obèses.

Extraits — `.dockerignore`

```
1 data/raw/
2 data/processed/
3 memoire/
4 documentation/
5 !data/processed/osm_nodes_with_routes.csv
6 !data/processed/atlas_routes_unified.csv
```

12.12. RECETTES DE DIAGNOSTIC

Commande

```
1 Ouvrir un shell dans le conteneur app
2 docker compose exec app bash
3
4 %# Tester la connexion MySQL depuis l'hôte
5 mysql -h 127.0.0.1 -P 3306 -u stops_user -p1234 -e "SHOW DATABASES;" 
6
7 %# Rebuilder l'image quand requirements.txt change
8 docker compose build app && docker compose up -d
```

12.13. RÉFLEXION

Ce qui marche bien. La séparation claire des rôles (db vs app), l'attente active sur MySQL et le *profil dev* évitent l'essentiel des écueils. Les drapeaux SKIP_DATA_IMPORT et MATCH_ONLY permettent de raccourcir radicalement les cycles de développement.

Aligner les permissions hôte/conteneur Pour éviter des soucis d'écriture avec le volume monté `./app`, on peut aligner l'UID/GID de l'utilisateur app du conteneur sur ceux de l'hôte via des *build args* :

Commande

```
1 docker compose build \\
2   --build-arg APP_UID=$(id -u) \\
3   --build-arg APP_GID=$(id -g) app
```

Dans `docker-compose.yml`, ces arguments sont déjà reliés à `APP_UID/APP_GID` (avec défaut

1000); ils peuvent aussi être définis dans un fichier .env.

Conclusion. Cette dockerisation est volontairement pragmatique : peu de magie, des scripts explicites, et des profils qui servent les usages quotidiens. En peaufinant la chaîne (build multi-étapes, non-root, healthchecks, secrets), on obtient une base de déploiement robuste.

CONCLUSION

Ce projet a cherché à approcher le problème de synchronisation des arrêts de transport public entre OSM et le système ATLAS en Suisse afin d'améliorer la précision des données de transport. Pour ce faire, nous avons mis en œuvre diverses méthodes de correspondance, telles que la correspondance exacte, par nom et par distance, tout en explorant la correspondance par itinéraire comme approche expérimentale. À ce jour, nous avons réussi à appairer 33 747 des 56 128 arrêts ATLAS. Par ailleurs, parmi les noeuds OSM sans correspondance actuelle, 27 759 sont associés à au moins un itinéraire, ce qui laisse entrevoir un potentiel significatif pour améliorer notre taux de correspondance grâce à l'intégration des données d'itinéraires.

Au-delà de ses aspects techniques, ce projet a constitué une expérience d'apprentissage particulièrement enrichissante. J'ai acquis des compétences en cartographie, dans les systèmes de routage des transports publics comme GTFS, ainsi qu'en développement web. Le défi de concevoir des algorithmes de correspondance et de rendre des données complexes accessibles m'a profondément stimulé, tout en mettant en lumière l'importance d'une communication claire dans les projets techniques. J'ai également trouvé fascinant d'explorer les cartes et de les examiner de près, car cela permet de mieux comprendre et apprécier le territoire suisse. Ce qui m'a le plus marqué, c'est le plaisir de découvrir ces domaines et de relever des défis qui, bien que complexes, se sont révélés passionnants.

En regardant vers l'avenir, plusieurs pistes d'amélioration se dessinent. On peut utiliser davantage d'informations disponibles, comme l'opérateur, parmi d'autres. Une analyse approfondie des balises et une importation prudente des noeuds OSM manquants s'imposent pour garantir l'intégrité des données. Un défi clé reste de déterminer, pour chaque correspondance, quel arrêt – OSM ou ATLAS – est le plus correct. Une solution envisagée serait d'améliorer notre application web et de l'ouvrir au public pour une vérification participative par des usagers familiers des zones concernées, ce qui nécessiterait d'importants efforts pour optimiser son ergonomie et ses fonctionnalités. Par ailleurs, la mise en place d'une structure de données robuste sera cruciale pour gérer et appliquer efficacement les corrections aux deux ensembles de données. Bien que le chemin à parcourir soit encore long, nos avancées actuelles renforcent notre détermination.

BIBLIOGRAPHIE

- Alembic Documentation.* URL : <https://alembic.sqlalchemy.org/> (visité le 09/08/2025).
- Amazon Simple Email Service (SES).* URL : <https://aws.amazon.com/ses/> (visité le 11/08/2025).
- Argon2 documentation.* URL : <https://argon2-cffi.readthedocs.io/en/stable/> (visité le 17/06/2025).
- ATLAS App SBB.* URL : <https://atlas.app.sbb.ch> (visité le 23/07/2025).
- Cloudflare Turnstile.* URL : <https://www.cloudflare.com/products/turnstile/> (visité le 16/06/2025).
- DE :Tag :highway=bus_stop.* URL : https://wiki.openstreetmap.org/wiki/DE:Tag:highway=bus_stop (visité le 14/07/2025).
- Docker Compose.* URL : <https://docs.docker.com/compose/> (visité le 15/08/2025).
- Docker Documentation.* URL : <https://docs.docker.com/> (visité le 12/08/2025).
- Environment variables with Compose.* URL : <https://docs.docker.com/compose/environment-variables/envvars/> (visité le 14/08/2025).
- Fernet (symmetric encryption).* URL : <https://cryptography.io/en/latest/fernet/> (visité le 18/06/2025).
- Flask Documentation.* URL : <https://flask.palletsprojects.com/> (visité le 05/08/2025).
- FR :Key :public_transport.* URL : https://wiki.openstreetmap.org/wiki/FR:Key:public_transport (visité le 16/07/2025).
- Jinja Documentation.* URL : <https://jinja.palletsprojects.com/> (visité le 06/08/2025).
- Key :local_ref.* URL : https://wiki.openstreetmap.org/wiki/Key:local_ref (visité le 19/07/2025).
- Leaflet Documentation.* URL : <https://leafletjs.com/> (visité le 07/08/2025).
- Liste des codes pays UIC.* URL : https://fr.wikipedia.org/wiki/Liste_des_codes_pays_UIC (visité le 02/08/2025).
- MySQL 8.0 Documentation.* URL : <https://dev.mysql.com/doc/> (visité le 16/08/2025).
- Open Transport Data Swiss.* URL : <https://opentransportdata.swiss/en/> (visité le 21/07/2025).
- Overpass Turbo.* URL : <https://overpass-turbo.eu/> (visité le 20/07/2025).
- Proposal :Public transport schema.* URL : https://wiki.openstreetmap.org/wiki/Proposal:Public_transport_schema (visité le 17/07/2025).
- Public transport.* URL : https://wiki.openstreetmap.org/wiki/Public_transport (visité le 15/07/2025).
- RFC 6238 : TOTP : Time-Based One-Time Password Algorithm.* URL : <https://www.rfc-editor.org/rfc/rfc6238> (visité le 15/06/2025).

SQLAlchemy Documentation. URL : <https://www.sqlalchemy.org/> (visité le 04/08/2025).
Traffic-points-actual-date. URL : <https://data.opentransportdata.swiss/en/dataset/traffic-points-actual-date> (visité le 01/08/2025).
wkhtmltopdf. URL : <https://wkhtmltopdf.org/> (visité le 13/08/2025).