

Yes. I checked the **actual Safe2Go architecture/documentation and database inventory**, rather than giving you a generic routing recommendation. There is a very clear path forward.

The good news is that **you have already built most of the foundation for an API-independent routing system**. The main problem is that the system is still treating each transport domain/region somewhat independently, so it cannot yet behave like a truly global Google/Waze-style journey engine.

Your current system already has about **505 GB of GTFS data, 586M stop-times, 28.7M trips, 5.4M stops, 315K routes and 113M precomputed route connections**. You also have 8 self-hosted Valhalla regions and substantial OSM/building/address data.

Safe2Go — Logic

My main recommendation

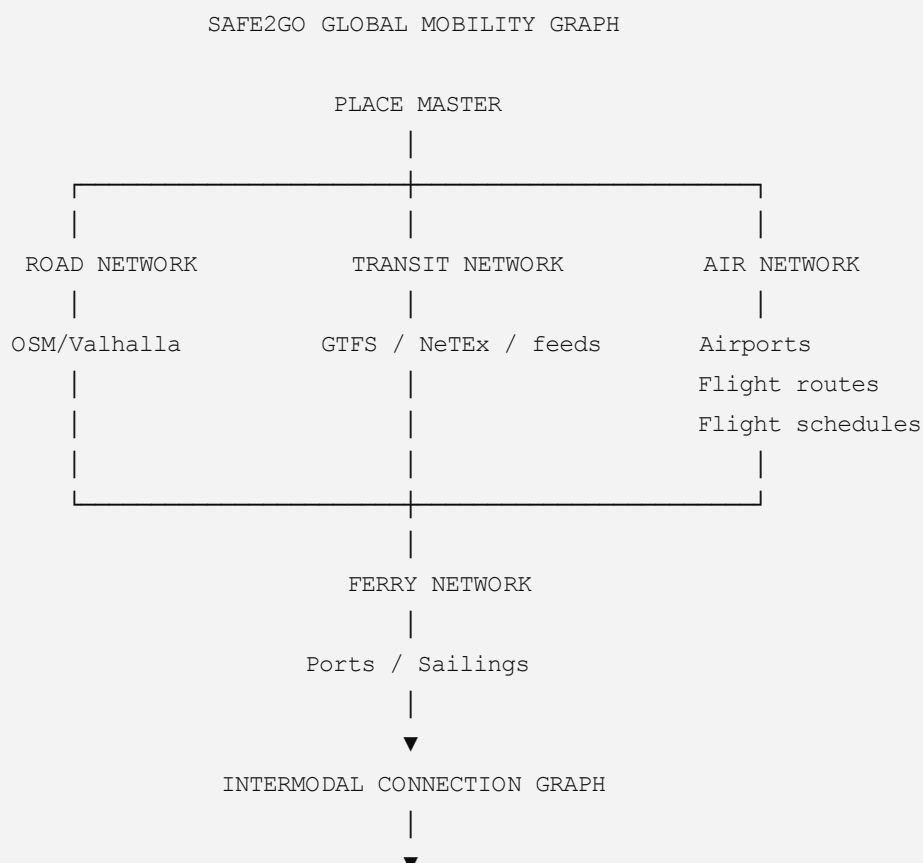
Don't try to remove APIs by simply replacing:

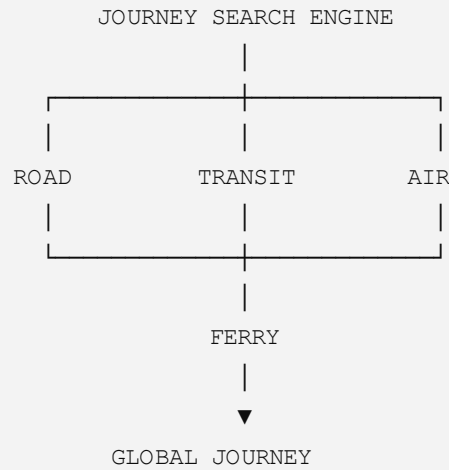
API A → Database A

API B → Database B

Instead, change Safe2Go into a **self-contained Global Mobility Graph**.

The architecture should become:





The biggest architectural problem today

Your eight Valhalla installations are currently **separate graphs**. Germany, India, USA, Europe, Africa, Australia, Canada and MSB do not connect to one another.

Safe2Go — Logic

That means:

Mannheim → Frankfurt

works.

Frankfurt → Singapore

can potentially work through your flight logic.

But:

Mannheim → Singapore → Kuala Lumpur → Bangkok

is not being treated as one continuous graph.

That's the fundamental thing I would fix.

1. Make Valhalla a regional road engine, not the global journey engine

I would not try to create one enormous global Valhalla graph.

That would be unnecessarily expensive and operationally difficult.

Keep regional Valhalla:

```
Valhalla Germany
Valhalla India
Valhalla Europe
Valhalla USA
Valhalla Canada
Valhalla Africa
Valhalla Australia
Valhalla MSB
...
```

Then introduce a layer above it:

```
Global Connector Graph
```

For example:

```
Mannheim
  |
  | road
  ▼
Frankfurt Airport
  |
  | flight
  ▼
Singapore
  |
  | MRT
  ▼
Singapore ferry/airport/rail nodes
```

The global engine doesn't care which Valhalla instance produced the road leg.

It only sees:

```
NODE → EDGE → NODE
```

2. Your biggest missing component: a proper universal node model

You need one common identifier system for:

```
Place
Address
City
Station
Airport
```

```
Port
Bus terminal
Bus stop
Rail station
Metro station
Border crossing
```

For example:

```
NODE

node_id
node_type
place_id
latitude
longitude
country
timezone
parent_city
parent_region
```

Then every transport system attaches to those nodes.

Example

```
FRA_AIRPORT
├─ road access
├─ S-Bahn
├─ ICE
├─ bus
├─ taxi
└─ flight network
```

That is what allows Safe2Go to make the jump between modes.

3. Flights are currently one of the largest holes

Your database already has **67,184 airport-pair routes**, which is useful. But those are primarily route relationships, not actual schedules.

Safe2Go — Logic

More importantly, the current logic checks essentially:

```
Origin airport → Destination airport
```

If there isn't a direct route, it stops.

The documentation explicitly identifies this as missing: connecting flights through hubs are not currently searched.

Safe2Go — Logic

Fix

Build:

Airport Graph

FRA

- └─ DXB
- └─ DOH
- └─ IST
- └─ SIN
- └─ ...

DXB

- └─ DEL
- └─ BOM
- └─ SIN
- └─ ...

Then run a **time-dependent graph search**.

So:

Mannheim

↓

FRA

↓

DOH

↓

DEL

↓

Delhi

becomes one journey.

4. But don't use OpenFlights as your final flight database

This is important.

Your current 67K flight routes come from OpenFlights, and your own documentation correctly identifies the weakness: it is historical/community-maintained and doesn't contain reliable current schedules or fares.

Safe2Go — Logic

So:

Keep it as a network seed.

But build:

```
Airport Master
+
Airline Master
+
Flight Route Master
+
Flight Schedule Master
+
Operating Calendar
+
Seasonality
```

The **schedule** is what the Journey Engine actually needs.

5. Trains and buses: you are actually much further ahead than you think

This is one of the strongest parts of Safe2Go.

You already have:

- 5.397M stops
- 28.7M trips
- 586M stop-times
- 314K routes
- 8,889 agencies
- 2,938 feeds
- 113M precomputed route connections.

Safe2Go — Logic

That's substantial.

The issue isn't "we need a better API."

The issue is **coverage**.

You need to systematically increase:

```
Country
↓
Operator
```

```
↓  
Feed  
↓  
Stops  
↓  
Trips  
↓  
Stop times  
↓  
Calendar
```

And when GTFS doesn't exist:

```
Official timetable  
↓  
Operator adapter  
↓  
Safe2Go normalized timetable
```

That solves India, parts of Africa, private buses, etc.

6. Your 113M `route_connections` table is a very good idea

I would actually take this further.

You currently have:

```
stop A → stop B  
trips/day  
sample time
```

Safe2Go — Logic

Turn that into a proper **Connection Graph**.

Something like:

```
connection_id  
from_node  
to_node  
service_id  
departure_time  
arrival_time  
operating_calendar  
mode  
operator
```

```
trip_id
transfer_group
source
confidence
valid_from
valid_to
```

Then your journey engine can search connections without repeatedly joining enormous GTFS tables.

That should make route planning dramatically faster.

7. You need a proper Transfer Graph

This is probably the most important missing piece after the global connector graph.

Suppose:

```
Train → Airport → Flight
```

You need to know:

```
Train arrival
  ↓
walk/transfer
  ↓
airport terminal
  ↓
minimum connection time
  ↓
flight departure
```

Likewise:

```
Flight → Train
Flight → Ferry
Ferry → Train
Bus → Flight
Metro → Train
```

Create:

```
intermodal_transfer
```

```
from_node
to_node
```

```
mode
distance
walking_time
minimum_transfer_time
maximum_transfer_distance
terminal
security_required
border_required
confidence
```

Now the engine knows whether a connection is physically possible.

8. This solves your intercontinental problem

For example:

Mannheim → Kathmandu

The engine could discover:

```
Mannheim
  ↓ train
Frankfurt
  ↓ airport transfer
FRA
  ↓ flight
DEL
  ↓ flight/bus
KTM
```

Or:

```
Mannheim
  ↓ train
Munich
  ↓ flight
Delhi
  ↓ flight
Kathmandu
```

Or another hub depending on schedules.

You don't program these combinations.

The graph discovers them.

9. Ferries should become exactly the same kind of graph

Instead of treating ferry as a special hard-coded list:

```
Genoa
Naples
Piraeus
...
```

which you currently do, create:

```
PORT
↓
FERRY SERVICE
↓
PORT
```

with:

```
departure
arrival
operating days
season
operator
vehicle allowed
passenger allowed
source
```

Your current architecture explicitly identifies ferry schedules as missing.

Safe2Go — Logic

Once they're normalized, ferry becomes just another edge in the global graph.

10. Road routing needs one major change

You currently have:

Valhalla → OSRM → straight-line fallback.

The documentation identifies OSRM public server as the external dependency outside your current eight regions.

Safe2Go — Logic

I would remove that.

Instead:

```
Country/region OSM PBF
    ↓
Valhalla tiles
    ↓
regional routing engine
```

Build Valhalla coverage progressively.

You already have map data for Argentina, Brazil, China, Europe, Mexico and Russia, but the application isn't even wired to all of those map regions yet.

Safe2Go — Logic

So there is low-hanging fruit here.

11. Don't build a single Valhalla for "Europe"

I'd actually refine this.

Use **logical routing regions**, but make them overlap at boundaries.

For example:

```
Western Europe
Central Europe
Eastern Europe
Nordics
Balkans
UK/Ireland
...
```

But each graph should have a **border/edge overlap**.

Then:

```
Germany graph
    ↓
Austria graph
    ↓
Italy graph
```

can be stitched through known border nodes.

That is better than attempting a giant global road graph.

12. Your address/search system is another API dependency you can remove

Your own documentation already has:

- 27.2M-row place search index
- OSM addresses
- localities
- India Post
- GeoNames
- building footprints
- locality aliases.

Safe2Go — Logic

That's the correct direction.

But I would change the search architecture to:

```
graph TD
    A[User input] --> B[Normalization]
    B --> C[Exact index]
    C --> D[Prefix index]
    D --> E[Token index]
    E --> F[Fuzzy index]
    F --> G[Spatial index]
    G --> H[Alias index]
    H --> I[Building/road/locality matching]
```

Not:

```
graph TD
    A[Search our DB] --> B[Photon]
    B --> C[Nominatim]
    C --> D[Mappls]
    D --> E[LocationIQ]
    E --> F[OpenCage]
```



The current fallback chain is still too dependent on external services.

Safe2Go — Logic

13. You specifically need prefix search

This addresses your earlier requirement:

user enters first letter → suggestions immediately

and then:

second letter → narrower suggestions.

Build PostgreSQL `pg_trgm` + prefix indexes, or preferably a dedicated local search index.

For example:

```
sa
↓
Sahibabad
Saharanpur
Saket
Salem
...
```

Then:

```
sat
↓
Satyam...
```

Then:

```
satyam e
↓
Satyam Enclave
```

This should be **local and sub-100 ms**, rather than waiting for an external geocoder.

14. GPS → address needs to be separated from search

Your GPS problem is slightly different.

Don't expect one database to magically know:

```
lat/lng
↓
Plot 130
Satyam Enclave
```

when no source contains that information.

Instead:

```
GPS
↓
nearest road
↓
nearest locality
↓
nearest building
↓
address evidence
↓
confidence
```

And return:

```
GPS position
Rajendra Nagar
Ghaziabad
Uttar Pradesh
201005

Address confidence: 0.62
```

rather than inventing:

Plot 130

This is particularly important in India.

15. You can remove Overpass completely

Your documentation already has Overpass as a last-resort stop lookup.

Safe2Go — Logic

You've already downloaded OSM.

So there is no reason to query Overpass during normal operation.

Use:

```
OSM PBF
↓
osm2pgsql
↓
PostGIS
↓
local spatial indexes
```

and make Overpass:

zero runtime dependency.

16. Traffic is the one major API dependency I would NOT try to eliminate immediately

Your architecture correctly identifies traffic as the hardest dependency.

Safe2Go — Logic

You can eliminate:

- Photon
- Nominatim
- Mappls
- Overpass
- Geoapify
- Foursquare
- OSRM public
- IP geolocation API

much more easily.

But real-time traffic is different.

You need millions of observations:

```
road segment
timestamp
speed
free-flow speed
direction
weather
incidents
```

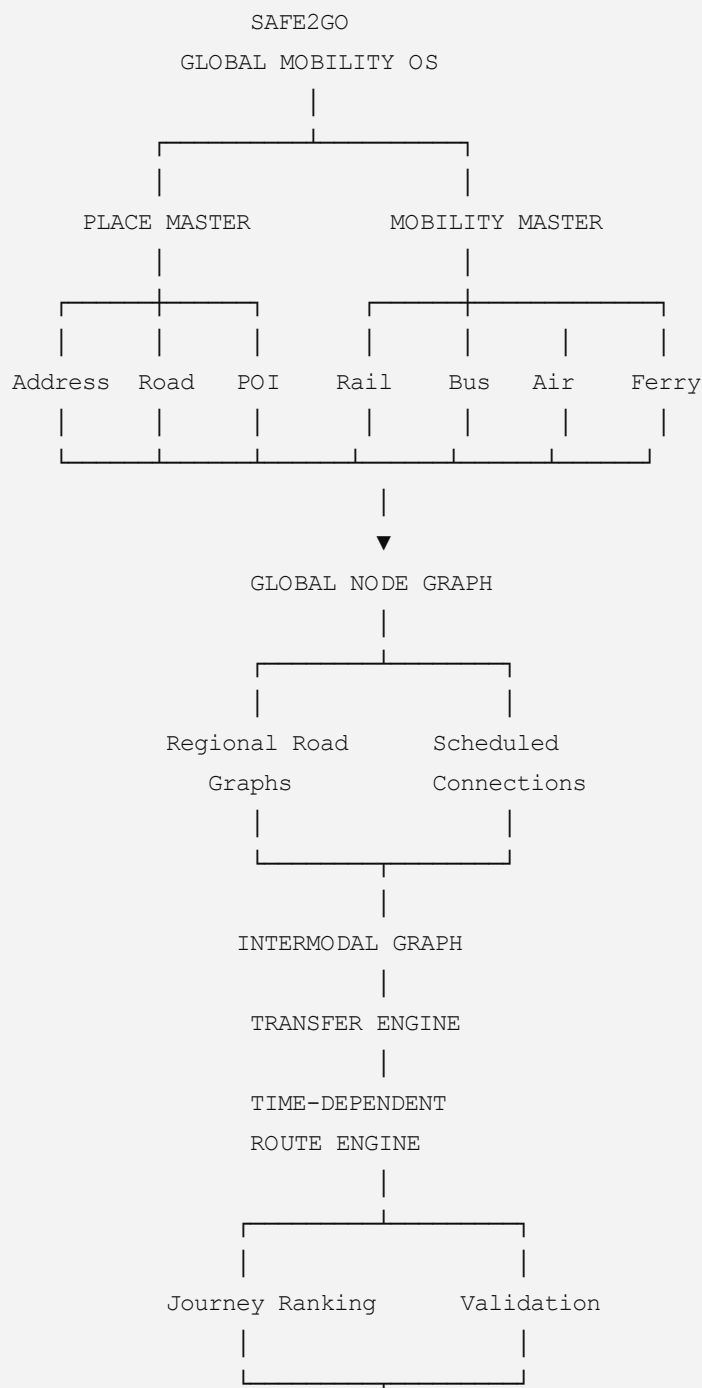
Eventually you can build your own traffic model from:

- Safe2Go GPS traces
- historical speeds
- road classification
- time/day
- weather
- incidents where legally sourced

But I would make this **Phase 4**, not block the API-removal project.

17. Your final architecture should look like this

This is what I would aim for:



What is missing today?

If I reduce the entire analysis to the important gaps, I get **10 items**:

Area	Current state	What is missing
Global road network	Partial	More Valhalla regions + stitching
Transit	Strong	More country/operator coverage
Address/search	Partial/strong foundation	Better India + worldwide locality/building coverage
Flights	Partial	Actual schedules + connecting-flight graph
Ferries	Weak	Operator adapters + schedule master
Bus/private bus	Partial	Non-GTFS operator adapters
Intermodal transfers	Weak	Universal transfer graph
Cross-border routing	Weak	Border nodes + country/region graph stitching
Multi-transfer journeys	Partial	Time-dependent multi-leg search
Live traffic	External	Own traffic model eventually

The existing documentation itself identifies multi-transfer transit, cross-region road routing, connecting flights, ferry schedules, uneven timetable coverage and traffic as outstanding areas.

Safe2Go — Logic

The most important change I would make

I would **stop thinking of Safe2Go as a collection of route planners**:

```
train planner
bus planner
flight planner
```

ferry planner
road planner

and make it:

one global time-dependent mobility graph with different data providers feeding it.

Then:

Flight
Train
Bus
Metro
Tram
Ferry
Car
Taxi
Walking
Cycling

are simply different **edge types**.

That gives you the permutations automatically.

And the API-removal target should be realistic

Remove completely

Photon → own search index

Nominatim runtime → own PostGIS

Mappls runtime → own India address/place data + aliases

Overpass → local OSM

Geoapify/Foursquare → own POI/station database

OSRM public → own Valhalla

ipwho.is → local GeoIP database

Your current replacement plan already points in this direction.

Safe2Go — Logic

Keep temporarily

GTFS-RT — because it is the operator's live data, not really a routing dependency.

Traffic provider — until Safe2Go has enough GPS history.

Commercial flight/ferry/bus APIs — only where you need current availability/pricing/booking.

Payments/auth/email/AI — these aren't routing dependencies and don't need to be eliminated.

The end-state

The user should be able to enter:

Plot 130, Satyam Enclave, Sahibabad → Kathmandu

and Safe2Go should do:

```
ADDRESS
↓
Local coordinates
↓
Nearest road / bus stop
↓
India road/transit network
↓
Delhi airport / railway / bus / etc.
↓
International connection
↓
Nepal network
↓
Kathmandu
↓
Final local road/transit
```

without calling a geocoder or routing API to figure out the basic route.

That's the architecture I would build toward. Your current database and infrastructure mean you're **much closer to this than starting from scratch**; the missing piece is mainly the **Global Connector + Intermodal Graph + schedule-aware journey engine**, not another API.

Sources

Your file library is full. New files won't be saved to your library, but you can still use them in this chat. To store new files for later, free up space or upgrade your storage.

[Rejoin Plus](#)

Manage

