



PROJET 6 : LE MODÈLE ORCA ASSISTÉ PAR L'IA : APPLICATION DANS UNE GARE FERROVIAIRE

Encadré par:
Mme. BANOUAR Oumayma

Présenté par:
CHAHBI ALAOUI HAYTHAM
ESSAHOUQUI ANAS
NAZIH REDA
SAMBA KHALIL
SELLAKH JAAFAR

PLAN

01

introduction

02

contextualisation

03

problématique

04

Théorie et
implementation en
Matlab

05

pipelines du projet

06

Data

07

développement des
modèles

08

Tuning des
paramètres

09

résultats et
perspectives

INTRODUCTION



CONTEXTUALISATION



World Cup 2030

PROBLÉMATIQUE

Déterminer comment placer un obstacle de manière optimale pour évacuer efficacement les personnes d'une salle de taille et de densité spécifiques ?

THÉORIE DU MODEL QUI GÈRE LA GESTION DES CONTACTES ET SON IMPLÉMENTATION EN MATLAB

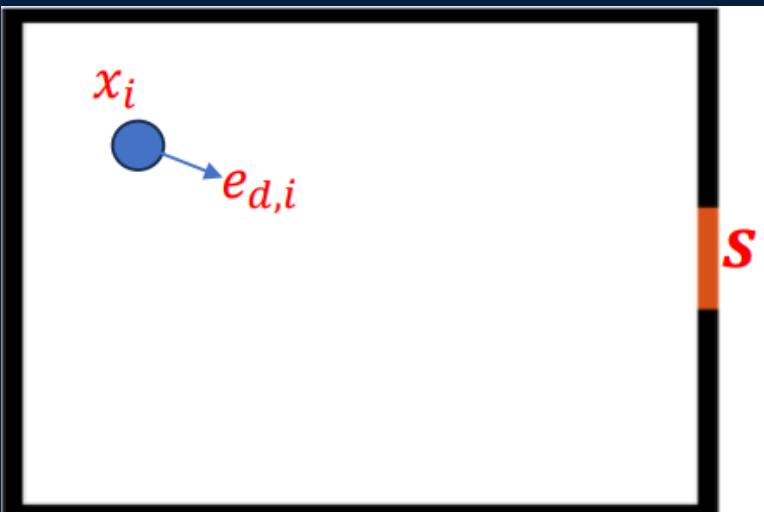
Introduction de la vitesse désirée:

Un agent est représenté par un disque de centre x_i et de rayon r

$$\frac{dx_i}{dt} = v_i, \quad i = 1 \dots n, \quad x_i(t=0) = x^0_i, \quad y_i(t=0) = y^0_i$$

$-x_i$: la position de l'agent i

$-v_i$: la vitesse réelle de l'agent i

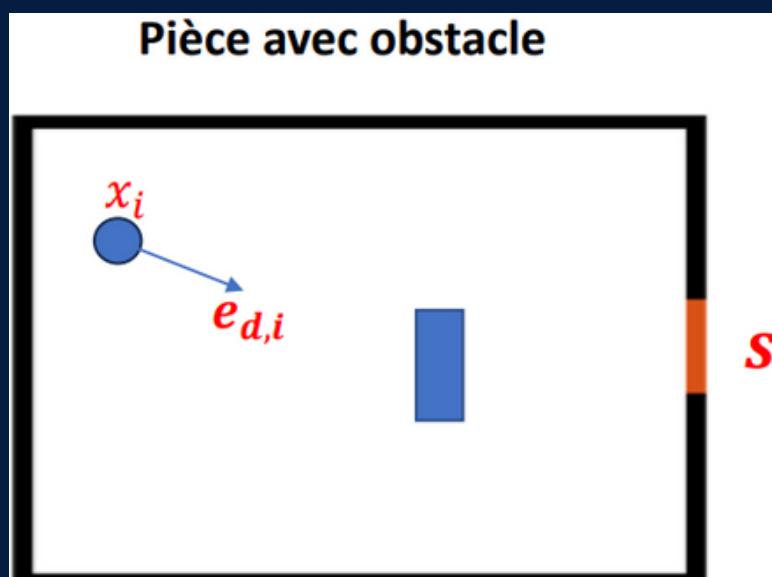


$$v_i = \|v_i\| \cdot e_{d,i}$$

- L'allure souhaitée $\|v_i\|$ converge vers $N(1.32; 0.26)$ loi normale
- Pièce sans obstacle : stratégie du chemin le plus court
- Pièce avec obstacle : Quand on ajoute des obstacles dans le domaine, comment calculer alors le chemin le plus court ?

$$e_{d,i} = \frac{s - x_i}{\|s - x_i\|}$$

La résolution de l'équation Eikonale !



L'Équation Eikonale et sa Contribution à la Gestion des Foules:

- **Définition :**

Une équation clé pour la propagation des ondes, où $\{ |\nabla u(x)| = F(x) \}$ lie le gradient de temps d'arrivée d'une onde ($u(x)$) à sa vitesse de propagation $F(x)$.

- **Ses applications dans la Gestion des Foules :**

Planification d'Évacuation: Utilisation pour simuler des mouvements de foule optimisés, minimisant les congestions lors d'évacuations d'urgence.

Simulation de Dynamiques de Foule: Aide à prévoir le flux des personnes en fonction de l'environnement, crucial pour la sécurité lors d'événements.

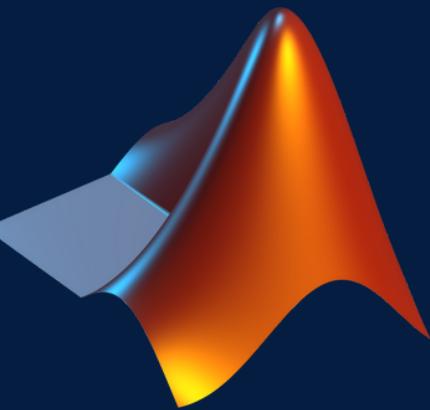
Optimisation de l'Espace : Informe la conception d'espaces pour améliorer le flux de circulation, réduisant les risques de bousculades.

Fast-Marching pour résoudre
l'équation Eikonale:

- Considérons le cas d'une situation d'évacuation normale i.e. chaque individu désire sortir « tranquillement » d'une salle contenant des obstacles.
- Pour définir la direction souhaitée pour chaque individu (le chemin le plus court pour atteindre la sortie) **On crée un champ de direction des vitesses souhaitées.**
- La méthode de **Fast Marching** permet de calculer la **distance géodésique $D(x)$** (distance minimale entre un point x de la pièce et la sortie.)
- Cette distance $D(x)$ vérifie une équation stationnaire dite **équation Eikonale :**

$$\begin{aligned} S(x) \|\nabla D(x)\| &= 1 && \text{sur } \Omega \\ D(x) &= g \text{ sur } && \partial\Omega \end{aligned}$$

Description de la fonction ‘Fast_Marching’:



[1]

Initialisation.m

Ajout obstacles/ murs & piétons

Boucle en temps :

Mise à jour de la position

Mise à jour de la vitesse

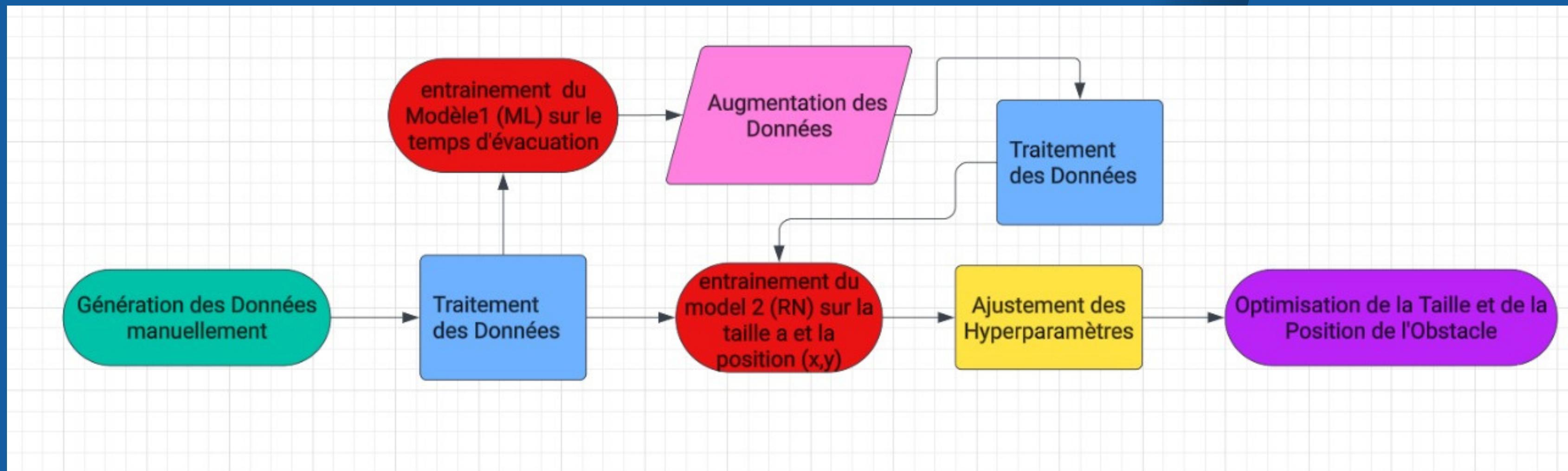
Affichage résultats

[2]

Fin boucle en temps :

PIPELINES DU PROJET

PIPELINE 1

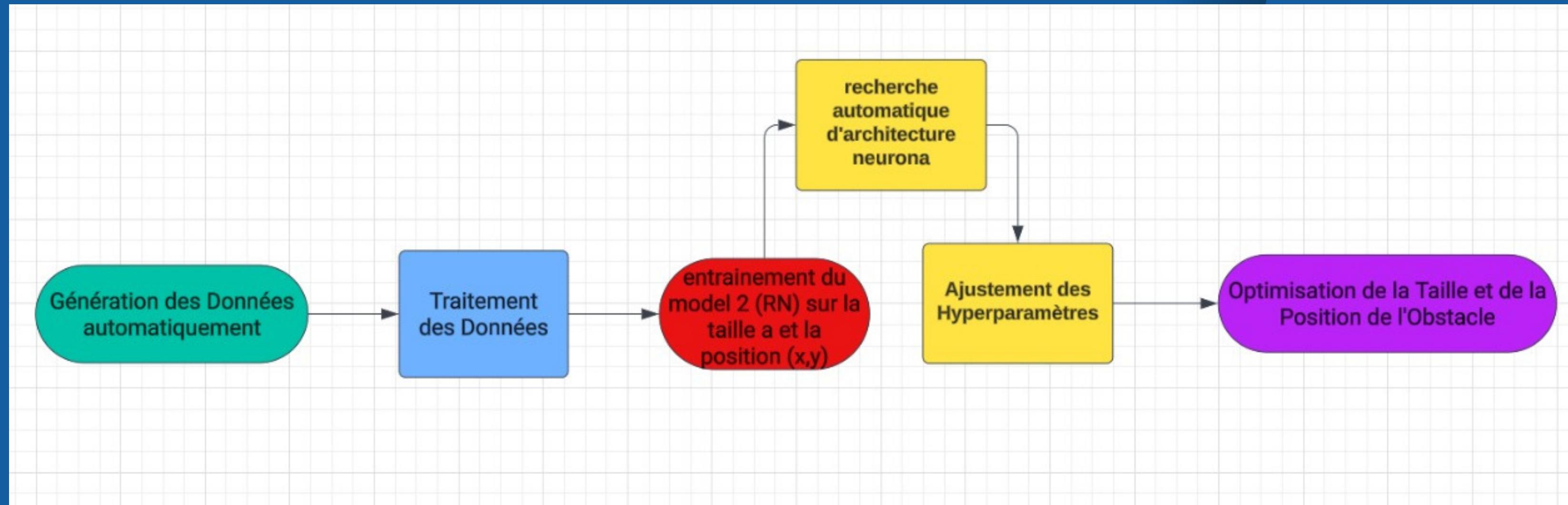


PIPELINE X

2

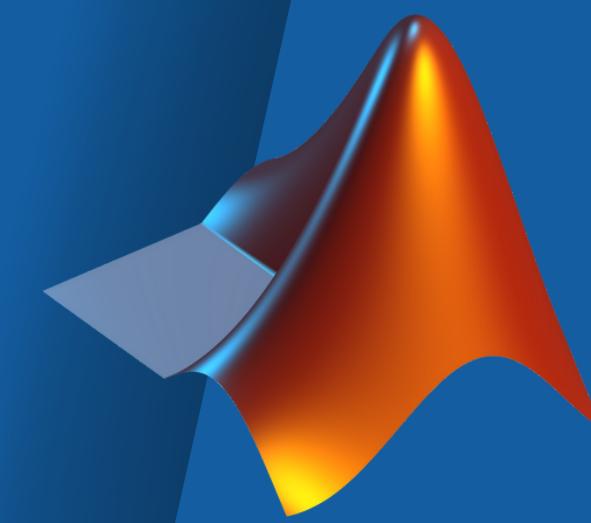
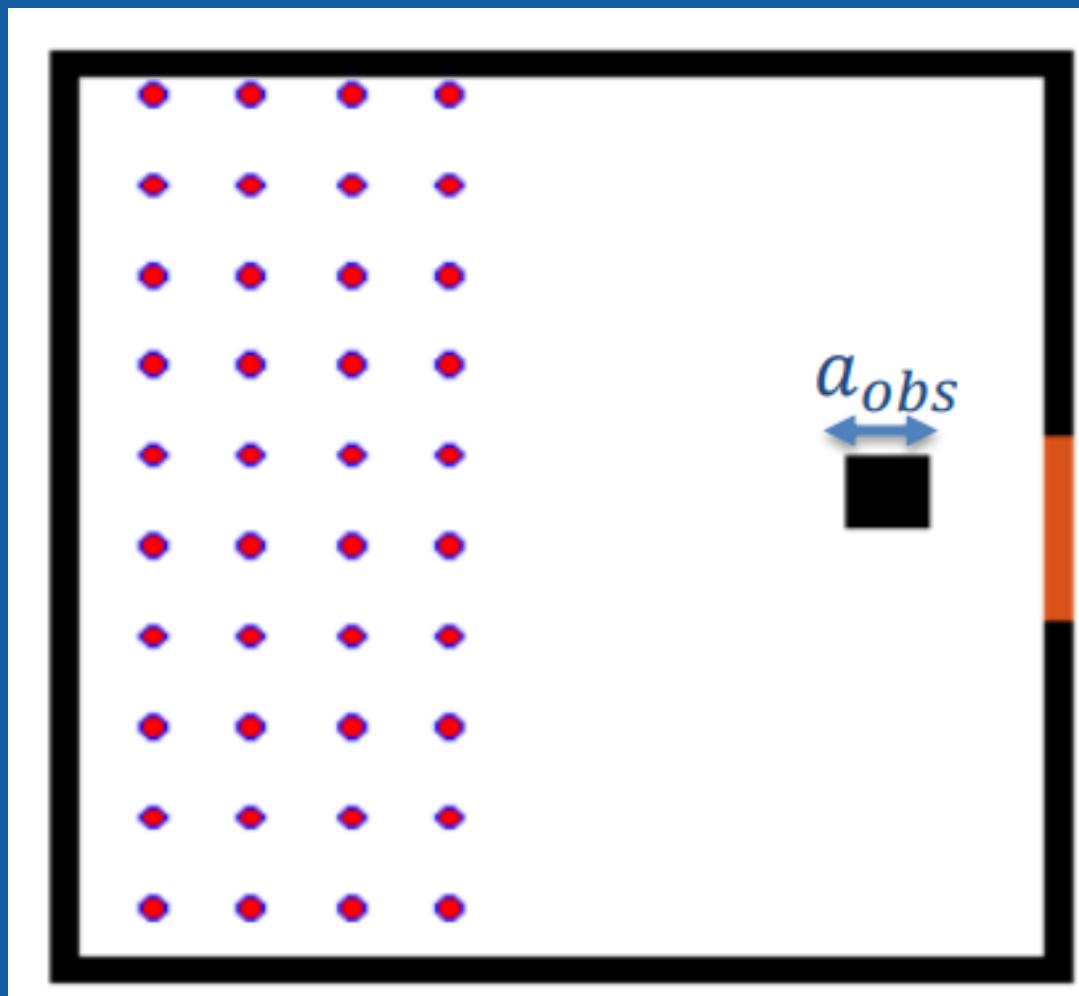
19

PIPELINE 2



DATA

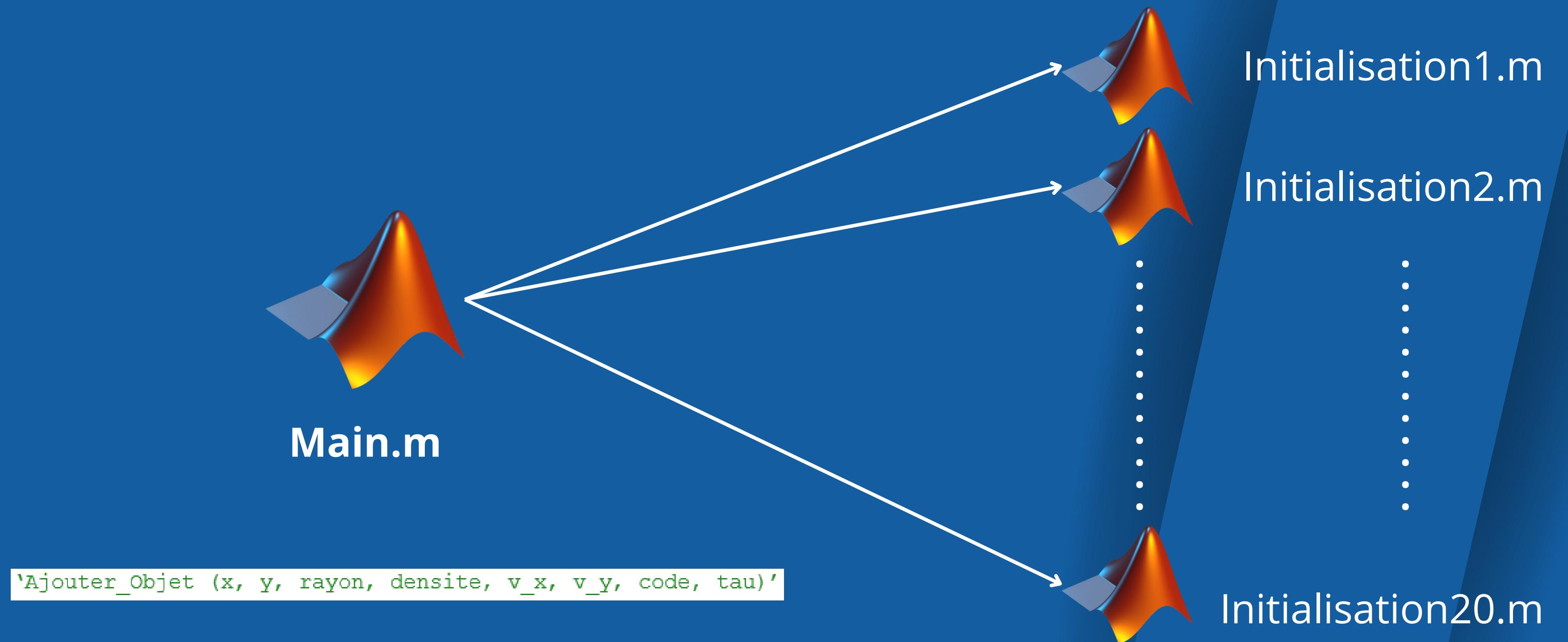
GÉNÉRATION DE LA DATA



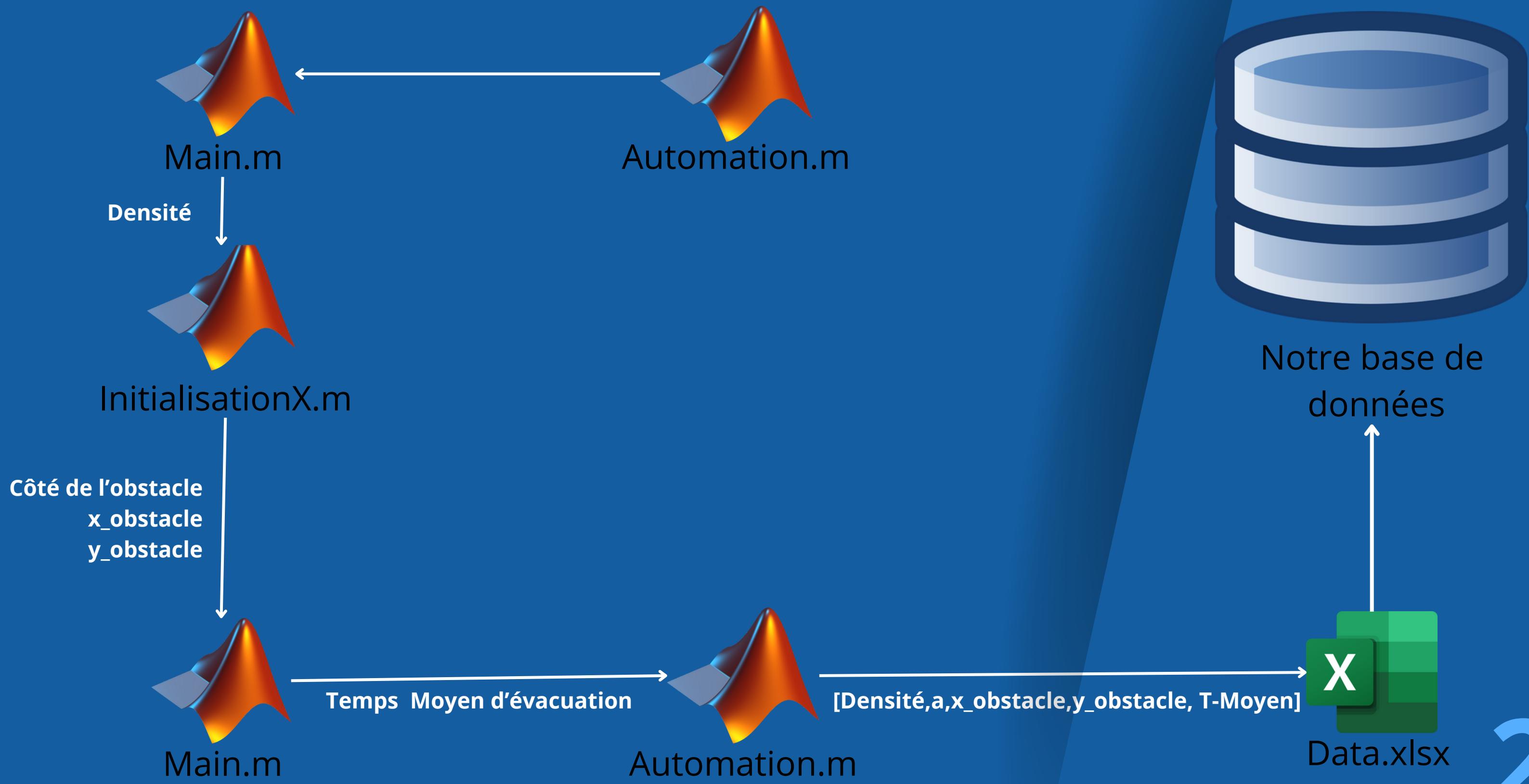
Initialisation.m

```
^ AJOUTER_OBSTACLE (X, Y, LONGUEUR, LARGEUR) ^
```

GÉNÉRATION DE LA DATA



GÉNÉRATION DE LA DATA



PRÉPARATION DE LA DATA

```
# Load the data into a pandas DataFrame
df = pd.read_csv('Test3.csv', delimiter=',')

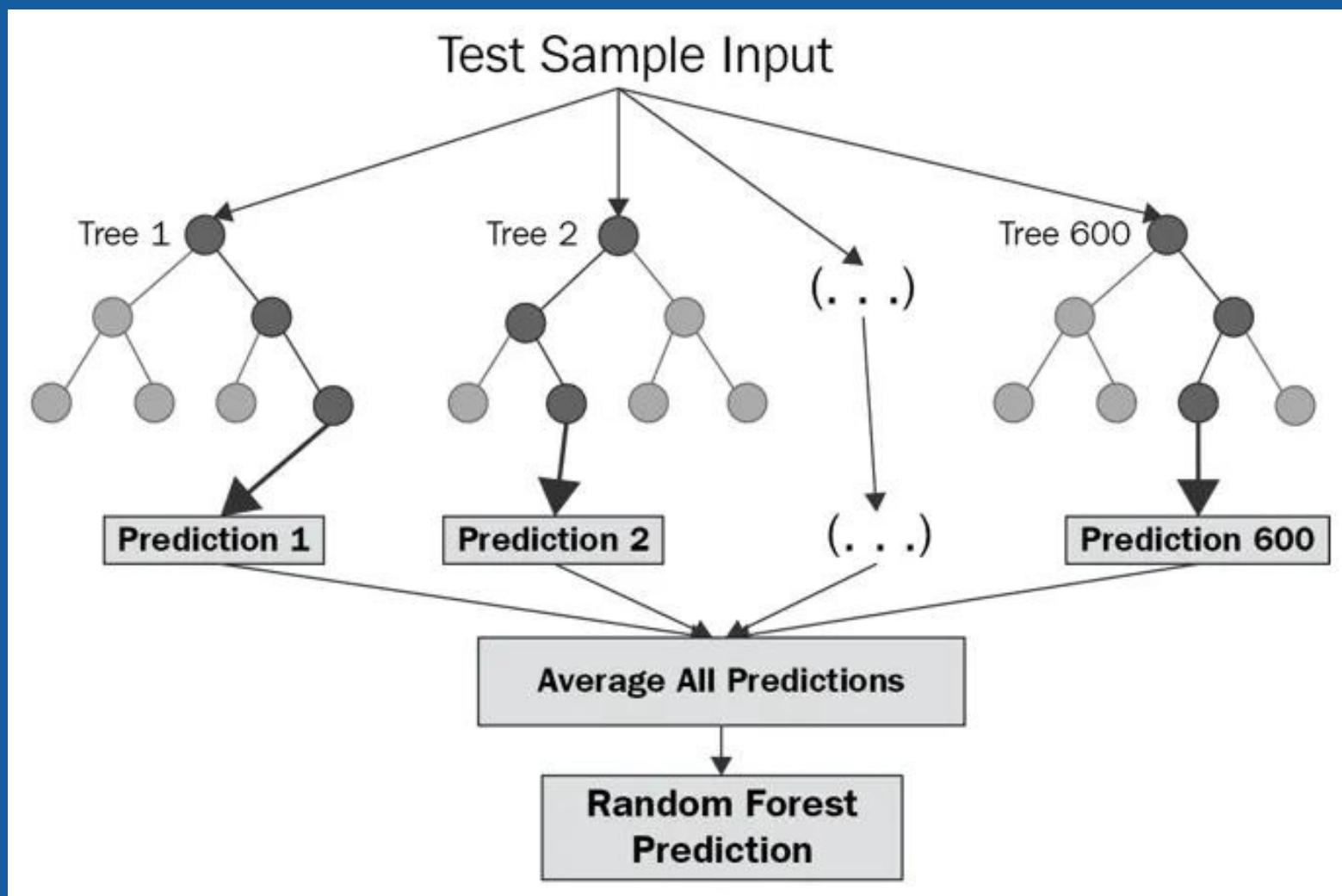
# For example, if there are missing values in 'Temps d'évacuation', we could drop them
df = df.dropna(subset=['T-Moyen'])

# Split the data into features and target variable
X = df[['Densité', 'T-Moyen', 'x_obstacle', 'y_obstacle']] # Features
y = df['a'] # Target variable

# Split the data into training set and test set (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

RANDOM FOREST



```
# Train a Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)
#model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

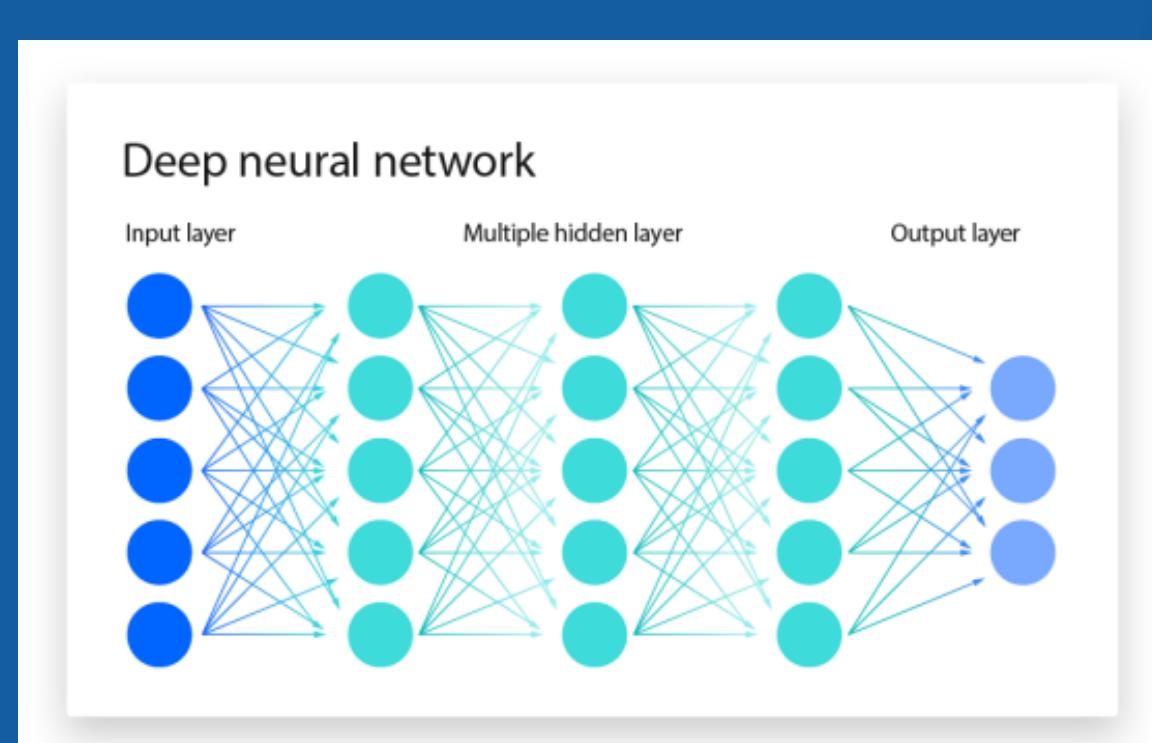
# Calculate relevant metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
predict = y_pred
```

RÉSULTATS

```
Mean Squared Error (MSE): 0.0401066887341772
Root Mean Squared Error (RMSE): 0.20026654422088877
Mean Absolute Error (MAE): 0.16201518987341768
R-squared (R2): -0.11224203914573971
```

NEURAL NETWORK

```
# Define the model architecture with increased complexity and L2 regularization
model = keras.Sequential([
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l1(0.01), input_shape=(4,)),
    layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l1(0.01)),
    #layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l1(0.01)),
    #layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(1)
])
```



RESULTATS

```
Mean Squared Error (MSE): 0.03612886419142171
Root Mean Squared Error (RMSE): 0.19007594322118124
Mean Absolute Error (MAE): 0.15911467731753481
R-squared (R2): -0.0019286769501680734
```

DÉVELOPPEMENT DES MODÈLES

TUNING DES HYPERPARAMETRES

grid search

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, scoring='r2', verbose=2)
grid_result = grid_search.fit(x_train_scaled, y_train)

# Get the best parameters and best score
best_params = grid_result.best_params_
best_score = grid_result.best_score_

print("Best parameters found: ", best_params)
print("Best R-squared score found: ", best_score)

# Use the best parameters to create and train the final model
final_model = create_model(regularizer_strength=best_params['regularizer_strength'])
final_model.fit(x_train_scaled, y_train, epochs=best_params['epochs'], batch_size=best_params['batch_size'], verbose=0)
```

```
Best parameters found: {'batch_size': 64, 'epochs': 100, 'regularizer_strength': 0.001}
Best R-squared score found: 0.000992091564614439
7/7 [=====] - 0s 2ms/step
```

NAS

```
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Dense(units=hp.Int('units', min_value=32, max_value=512, step=32),
                           activation='relu', input_shape=(4,)))
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(layers.Dense(units=hp.Int('units_' + str(i), min_value=32, max_value=512, step=32),
                               activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Instantiate the tuner and perform the search
tuner = RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=5, # Adjust the number of trials as needed
    executions_per_trial=3,
    directory='NAS',
    project_name='test_nas'
)

tuner.search(X_train_scaled, y_train, epochs=100, validation_data=(X_test_scaled, y_test))
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	160
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 32)	1056
dense_3 (Dense)	(None, 32)	1056
dense_4 (Dense)	(None, 1)	33
=====		
Total params:	3,361	
Trainable params:	3,361	
Non-trainable params:	0	

RÉSULTATS ET PERSPECTIVES

ENTRAINEMENT SUR A

random regression model

RMSE=0.1693
R2= 0.24

after Grid search

after NAS

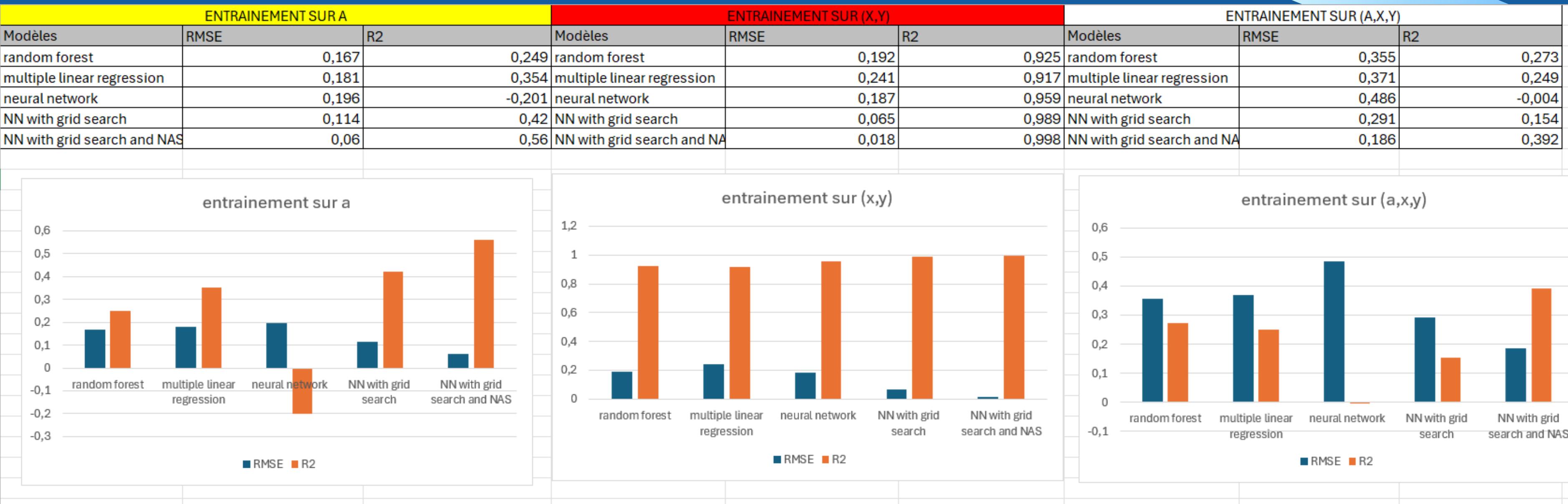
neural network model

RMSE=0.1146
R2=0.42

RMSE=0.08
R2=0.49

RMSE=0.1967
R2=-0.202

RESULTATS



PERSPECTIVE

CONCLUSION

Merci Pour Votre
Attention !



PROJET 6 : LE MODÈLE ORCA ASSISTÉ PAR L'IA : APPLICATION DANS UNE GARE FERROVIAIRE

Présentée par:
CHAHBI ALAOUI HAYTHAM
ESSAHOUQUI ANAS
NAZIH REDA
SAMBA KHALIL
SELLAKH JAAFAR