

# *TUTORIEL*

## Embarquer un modèle d'IA sur STM32

Pierre CHOUTEAU & Elisa DELHOMMÉ

# Objectif du tutoriel

L'embarquement d'un modèle d'intelligence artificielle sur une carte électronique comme la STM32 peut s'avérer utile dans de nombreux projets. Celui-ci donne la possibilité de faire tourner un modèle sur la carte directement, et non sur un serveur tel qu'un GPU. Outre la nécessité d'embarquement dans certains cas, cela permet des économies au niveau de la consommation d'énergie. Sujet d'ailleurs très discuté en ce moment dans le monde de l'intelligence artificielle.

Ce tutoriel a été réalisé à la suite d'un projet en école d'ingénieurs et suite au manque d'explications constaté à ce sujet.

Pour l'exemple, il s'inscrit dans le cadre d'un **modèle simple de prédiction de sinus**.

# Nécessaire préalable au suivi du tutoriel



## Modèle Keras

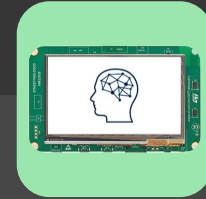
En format .h5 ou .tflite  
(les deux fonctionnent  
exactement pareil)

**Ici:** .tflite issu de  
l'entraînement d'un modèle  
de prédiction de sinus

*Notebook et .h5 disponibles  
sur le GitHub !*



## STM32 CubeIDE avec son extension X-Cube-AI



## Carte électronique

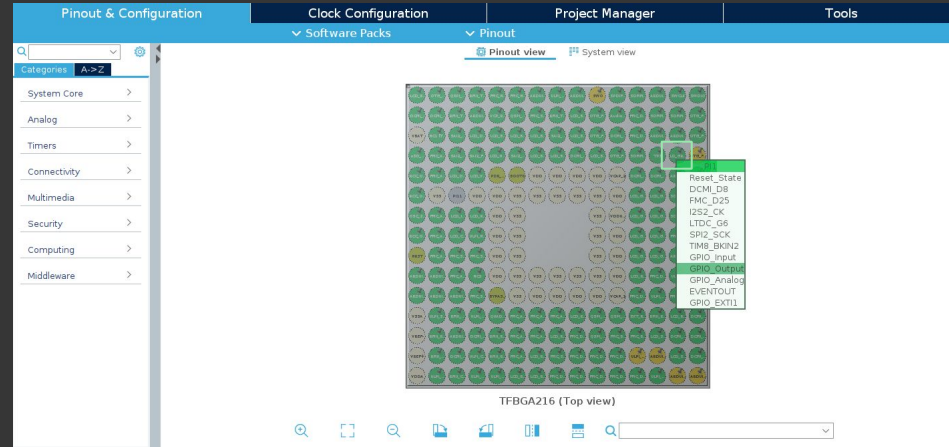
**Ici:**  
STM32F746NG-Discovery

- ➔ La version de CubeIDE utilisée est la **1.7.0** (1.8.0 fonctionne également)  
**ATTENTION :** Les versions antérieures peuvent ne pas convenir à ce tuto !
- ➔ Le firmware F7 utilisé est le **1.16.1**.

# Étape 1 - Préparation 1/2

## Création du projet et initialisation

La toute première étape consiste en la création d'un projet sur le logiciel *STM32CubeIDE* et l'initialisation des ports. En particulier, on initialisera le port GPIO PI1 afin d'utiliser la LED verte pour confirmer le bon fonctionnement du code par la suite.



# Étape 1 - Préparation 2/2

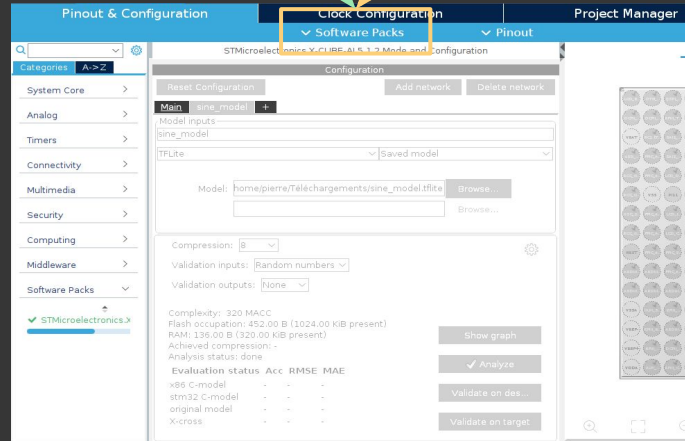
## Téléchargement de l'extension X-Cube-AI et choix de la version

L'extension *X-Cube-AI* est celle qui nous permettra de convertir le modèle *Keras* en code en langage *C* afin d'être exécutable sur la STM32.

Les versions de cette extension n'étant pas rétro-compatible, on sélectionne la version **5.1.2** pour le bon déroulé de la suite. (Un passage sur une autre version peut ne pas fonctionner !)

Installation de X-Cube-AI

- > Software Packs
- > Manage Software Packs
- > STMicroelectronics
- > Installer la version souhaitée



## Sélection de X-Cube-AI

- > Software Packs
- > Select components
- > X-Cube-AI
- > Cocher 5.1.2

(Si la version ne s'y trouve pas:  
Désinstaller et réinstaller la  
version résout le problème)

Interface d'accueil de STM32Cube IDE

# Étape 2 - Import du modèle

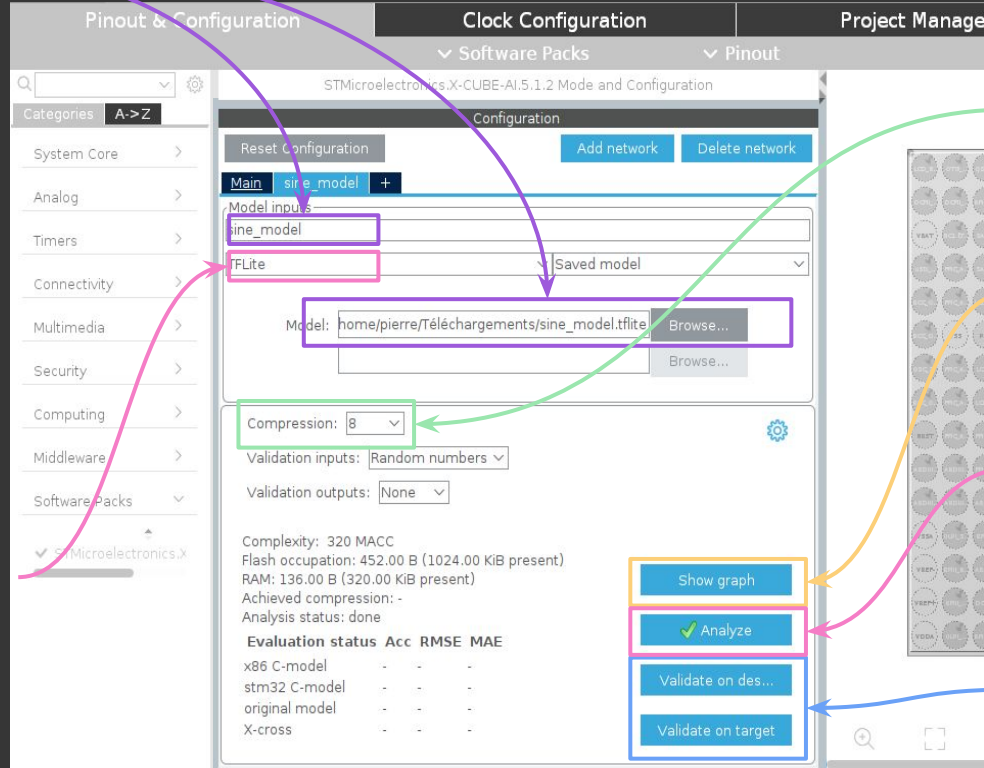
## Import du modèle

Sélectionner le modèle à importer et lui donner un nom (à mémoriser car celui-ci sera employé dans le code généré en C)

## Sélection du type de modèle

Sélectionner le format du fichier importé

- TFlite pour un .tflite
- Keras pour un .h5



## Compression des poids

Celle-ci n'est pas obligatoire mais peut s'avérer utile, voire nécessaire, dans le cas de modèle lourd (avec des couches récurrentes par exemple)

## Visualisation du modèle (optionnel)

## Analyse

Permet l'obtention d'informations relatives à la consommation en mémoire

## Validation du modèle

Tester la bonne validation du modèle sur le bureau puis sur la carte

Une fois ces étapes effectuées, il suffit d'enregistrer (CTRL+S) afin de générer le code en C.

## Étape 3 - Ajustements

Le code C est alors généré par l'extension et il ne doit pas être modifié. La génération permet de faire tourner le modèle et de le faire fonctionner. Cependant, il est nécessaire que l'utilisateur écrive lui-même la partie du code permettant d'initialiser le modèle, de faire une prédiction et d'ajuster cette partie au processus qu'il souhaite réaliser.

Quoi qu'il arrive, il faut écrire du code uniquement dans les parties délimitées par les mentions **USER CODE BEGIN** et **USER CODE END**. Du code écrit en dehors risque d'être supprimé par la génération d'un nouveau code.

Dans l'exemple de la prédiction de sinus, il est seulement nécessaire de fournir un nombre de type *float* au modèle, et de le récupérer grâce aux *buffers* introduits automatiquement par la conversion.

Incrément du test pour tester plusieurs valeurs

```
// Create instance of neural network
ai_err = ai_sine_model_create(&sine_model, AI_SINE_MODEL_DATA_CONFIG);
if (ai_err.type != AI_ERROR_NONE)
{
    buf_len = sprintf(buf, "Error: could not create NN instance\r\n");
    HAL_UART_Transmit(&uart6, (uint8_t *)buf, buf_len, 100);
    while(1);
}
```


```
// Initialize neural network
if (!ai_sine_model_init(sine_model, &ai_params))
{
    buf_len = sprintf(buf, "Error: could not initialize NN\r\n");
    HAL_UART_Transmit(&uart6, (uint8_t *)buf, buf_len, 100);
    while(1);
}
```

```
// Fill input buffer (use test value)
for (uint32_t i = 0; i < AI_SINE_MODEL_IN_1_SIZE; i++)
{
    ((ai_float *)in_data)[i] = (ai_float)test;
}
test += 3.1415 / 2;
```

Code complet disponible sur le GitHub !


Dans d'autres cas nécessitant un traitement des données en amont et/ou en aval du modèle, il peut être nécessaire de le réaliser en langage C dans ces parties.

## Étape 4 - Test & Run


Dans notre cas (de prédiction de sinus et où l'écran de la STM32 n'est pas initialisé), on peut voir les résultats de la prédiction du modèle dans le mode debugger .

La prédiction de la valeur du sinus renseignée dans la variable `test` est visible dans la variable `y_val`.

Par exemple, pour une valeur 

 <code>test</code>	float	4.71225023
---	-------	------------

 ( $\approx 3\pi/2$ ),  
on obtient 

 <code>y_val</code>	float	-1.05025923
--	-------	-------------

 et la prédiction est réussie !