Отчёт

Маевский Илья

Группа МИК22

Цель проекта:

Интегрировать языковую модель в игровой движок Unity.

Практическая часть:

Для интеграции готовой языковой модели в игровой движок Unity будем использовать такой инструмент как Unity Sentis. Он позволяет разработчикам интегрировать функции искусственного интеллекта в игры и приложения без необходимости использования внешних серверов или инструментов.

Для начала работы с Sentis его необходимо установить в Unity. Опишем шаги по установке:

1. Создайте новый проект Unity или откройте существующий;
2. чтобы открыть менеджер пакетов, перейдите в "Окно" > "Менеджер пакетов";
3. нажмите "+" и выберите "Добавить пакет по имени"...;
4. введите Com.unity.sentis;
5. нажмите "Добавить", чтобы добавить пакет в свой проект.

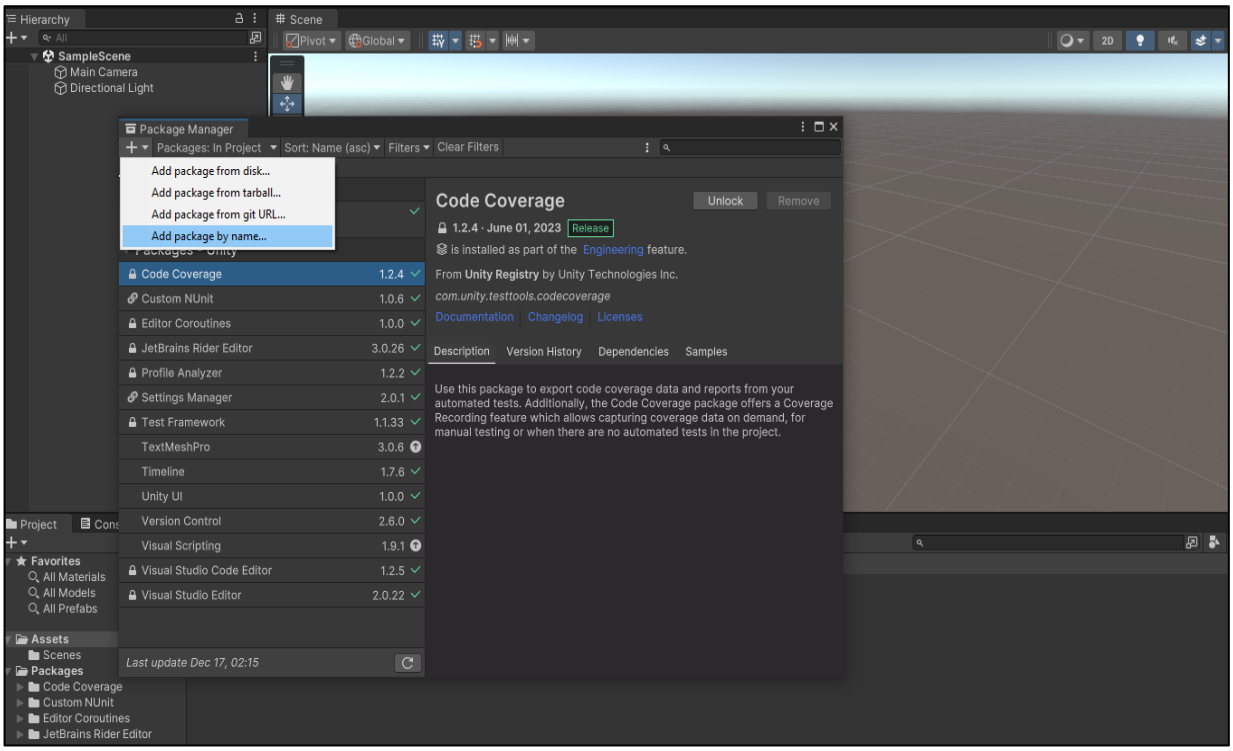

Рисунок 1 — Установка пакета Unity Sentis в Unity

Импорт модели и обязательных файлов

После завершения импорта пакета в наш проект, скачиваем необходимую нейронную модель в формате sentis или ONNX, файлы merges.txt и vocab.json с сайта huggingface и переносим их в проект в папку Assets/StreamingAssets, так же заранее создадим папку Scripts где будут хранится будущие скрипты. Для данной работы я выбрал небольшую языковую модель Tiny stories, которую специально подготовили сами Unity для работы с Sentis.

Создание скрипта для взаимодействия с нейронной сетью

Взаимодействие с моделью должно быть реализовано через UI интерфейс с InputField куда пользователь будет записывать свой промпт, Text в котором будет записываться сгенерированный текст модели и Button -

кнопка, при нажатии на которую промпт будет отправляться модели для генерации.

Так же учтём, что при повторном нажатии на кнопку, должна начаться новая генерация по новому промпту, записанном в InputField.

Создадим скрипт RunTinyStories.cs в папке Scripts. Первым делом подключим необходимые библиотеки:

using System.Collections.Generic; - библиотека для работы с коллекциями (списки, словари и т. д.).

using UnityEngine; - основная библиотека Unity для работы с игровыми объектами, компонентами, физикой и т. д.

using Unity.Sentis; - библиотека для запуска предобученных нейросетей (ONNX-моделей) прямо в Unity.

using System.IO; - библиотека для чтение/запись файлов (например, merges.txt, vocab.json или весов модели).

using System.Text; - библиотека для работы с кодировками и текстом (например, UTF-8 для BPE-токенизации).

using FF = Unity.Sentis.Functional; - упрощённый API для операций с тензорами в Sentis (аналог функционала из PyTorch).

using TMPro; - библиотека для продвинутого отображения текста в Unity (поддержка Unicode, шрифты).

using UnityEngine.UI; - библиотека для работы с UI-элементами (кнопки, слайдеры, Canvas).

Далее объявляем необходимые поля

```
const BackendType backend = BackendType.GPUCompute;

string outputString = "";
const int maxTokens = 100;
const float predictability = 5f;
const int END_OF_TEXT = 50256;

string[] tokens;
IWorker engine;

int currentToken = 0;
int[] outputTokens = new int[maxTokens];

bool runInference = false;
const int stopAfter = 100;
int totalTokens = 0;

string[] merges;
Dictionary<string, int> vocab;

public TMP_Text outputText;
public TMP_InputField inputField;
public Button generateButton;

int[] whiteSpaceCharacters = new int[256];
int[] encodedCharacters = new int[256];
```

Рисунок 2 — Объявление переменных скрипта RunTinyStories.cs

Теперь напишем все необходимые методы для работы скрипта:

InitializeModel:

- Освобождает предыдущую модель (engine.Dispose())
- Загружает ONNX-модель (tinystories.sentis)
- Компилирует её с помощью Functional (FF) для работы с токенами
- Создаёт "движок" для инференса через WorkerFactory

OnGenerateButtonClicked:

- Сбрасывает состояние генерации (ResetGenerationState)
- Инициализирует модель (InitializeModel)
- Начинает генерацию с текста из inputField

RunInference:

- Передаёт текущие токены в модель (engine.Execute)
- Получает предсказание следующего токена (Multinomial)
- Обновляет выходной текст, проверяя на END_OF_TEXT или лимит токенов

DecodePrompt:

- Конвертирует текст в токены (GetTokens)
- Заполняет буфер outputTokens начальными токенами

LoadVocabulary:

- Загружает vocab.json (словарь токенов) и merges.txt (BPE-правила)
- Создает обратный словарь tokens для декодирования

GetUnicodeText/GetASCIIText:

Конвертирует текст между UTF-8 и ASCII-подобной кодировкой (для обработки спецсимволов).

ShiftCharacterDown/ShiftCharacterUp:

Заменяет спецсимволы (например, пробелы) на числа >256 и обратно.

SetupWhiteSpaceShifts/IsWhiteSpace:

Настраивает таблицы замены для пробельных символов.

GetTokens/ApplyMerges:

- Разбивает текст на токены с помощью BPE (правила из merges.txt)
- Конвертирует токены в ID через vocab.json

ResetGenerationState:

Сбрасывает буферы (outputTokens, outputString), счётчики и освобождает модель.

OnDestroy:

Освобождает ресурсы модели при закрытии сцены.

```
bool IsWhiteSpace(int i)
{
    return i <= 32 || (i >= 127 && i <= 160) || i == 173;
}


List<int> GetTokens(string text)
{
    text = GetASCIIText(text);

    var inputTokens = new List<string>();
    foreach (var letter in text)
    {
        inputTokens.Add(letter.ToString());
    }

    ApplyMerges(inputTokens);

    var ids = new List<int>();
    foreach (var token in inputTokens)
    {
        if (vocab.TryGetValue(token, out int id))
        {
            ids.Add(id);
        }
    }

    return ids;
}
```

Рисунок 3 — методы скрипта RunTinyStories.cs

```
Ссылок: 1
void ApplyMerges(List<string> inputTokens)
{
    foreach (var merge in merges)
    {
        string[] pair = merge.Split(' ');
        int n = 0;
        while (n >= 0)
        {
            n = inputTokens.IndexOf(pair[0], n);
            if (n != -1 && n < inputTokens.Count - 1 && inputTokens[n + 1] == pair[1])
            {
                inputTokens[n] += inputTokens[n + 1];
                inputTokens.RemoveAt(n + 1);
            }
            if (n != -1) n++;
        }
    }
}

Ссылок: 1
void ResetGenerationState()
{
    if (engine != null)
    {
        engine.Dispose(); // Удаляем старый движок
    }

    currentToken = 0;
    totalTokens = 0;
    outputTokens = new int[maxTokens];
    outputString = "";
}

Сообщение Unity | Ссылок: 0
private void OnDestroy()
{
    engine?.Dispose();
}
```

Рисунок 4 — методы скрипта RunTinyStories.cs

```
Ссылок: 1
void InitializeModel()
{
    engine?.Dispose(); // Удаляем предыдущий движок, если он существует
    var model1 = ModelLoader.Load(Path.Join(Application.streamingAssetsPath, "tinystories.sentis"));
    var model2 = FF.Compile(
        (input, currentToken) =>
        {
            var row = FF.Select(model1.Forward(input)[8], 1, currentToken);
            return FF.Multinomial(predictability * row, 1);
        },
        (model1.inputs[0], InputDef.Int(new TensorShape()))
    );

    engine = WorkerFactory.CreateWorker(backend, model2);
}

Ссылок: 1
void OnGenerateButtonClicked()
{
    ResetGenerationState();  // Полностью сбрасываем состояние перед запуском новой генерации
    InitializeModel();       // Создаём новый экземпляр модели
    outputString = inputField.text;
    DecodePrompt(outputString);
    runInference = true;
}
```

Рисунок 5 — методы скрипта RunTinyStories.cs

```
Ссылок: 1
void RunInference()
{
    using var tokensSoFar = new TensorInt(new TensorShape(1, maxTokens), outputTokens);
    using var index = new TensorInt(currentToken);

    engine.Execute(new Dictionary<string, Tensor> { { "input_0", tokensSoFar }, { "input_1", index } });

    var probs = engine.PeekOutput() as TensorInt;
    probs.CompleteOperationsAndDownload();

    int ID = probs[0];

    if (currentToken >= maxTokens - 1)
    {
        for (int i = 0; i < maxTokens - 1; i++) outputTokens[i] = outputTokens[i + 1];
        currentToken--;
    }

    outputTokens[++currentToken] = ID;
    totalTokens++;

    if (ID == END_OF_TEXT || totalTokens >= stopAfter)
    {
        runInference = false;
    }
    else
    {
        outputString += GetUnicodeText(tokens[ID]);
    }

    outputText.text = outputString;
}
```

Рисунок 6 — методы скрипта RunTinyStories.cs

```
void DecodePrompt(string text)
{
    var inputTokens = GetTokens(text);

    for (int i = 0; i < inputTokens.Count; i++)
    {
        outputTokens[i] = inputTokens[i];
    }
    currentToken = inputTokens.Count - 1;
}

Ссылок: 1
void LoadVocabulary()
{
    var jsonText = File.ReadAllText(Path.Join(Application.streamingAssetsPath, "vocab.json"));
    vocab = Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string, int>>(jsonText);
    tokens = new string[vocab.Count];
    foreach (var item in vocab)
    {
        tokens[item.Value] = item.Key;
    }

    merges = File.ReadAllLines(Path.Join(Application.streamingAssetsPath, "merges.txt"));
}

Ссылок: 1
string GetUnicodeText(string text)
{
    var bytes = Encoding.GetEncoding("ISO-8859-1").GetBytes(ShiftCharacterDown(text));
    return Encoding.UTF8.GetString(bytes);
}

Ссылок: 1
string GetASCIIText(string newText)
{
    var bytes = Encoding.UTF8.GetBytes(newText);
    return ShiftCharacterUp(Encoding.GetEncoding("ISO-8859-1").GetString(bytes));
}
```

Рисунок 6 — методы скрипта RunTinyStories.cs

```
Ссылок: 1
string ShiftCharacterDown(string text)
{
    string outText = "";
    foreach (char letter in text)
    {
        outText += ((int)letter <= 256) ? letter :
            (char)whiteSpaceCharacters[(int)(letter - 256)];
    }
    return outText;
}

Ссылок: 1
string ShiftCharacterUp(string text)
{
    string outText = "";
    foreach (char letter in text)
    {
        outText += (char)encodedCharacters[(int)letter];
    }
    return outText;
}

Ссылок: 1
void SetupWhiteSpaceShifts()
{
    for (int i = 0, n = 0; i < 256; i++)
    {
        encodedCharacters[i] = i;
        if (IsWhiteSpace(i))
        {
            encodedCharacters[i] = n + 256;
            whiteSpaceCharacters[n++] = i;
        }
    }
}
```

Рисунок 7 — методы скрипта RunTinyStories.cs

Вызываем методы в стандартных методах Unity (Start и Update)

С полным скриптом можно ознакомиться в приложении а.

Проверка работы

Сохраняем скрипт и перетащим его на Main Camera (или любой другой объект на сцене). Создаём Canvas и в нём объекты Button(TMP), InputField(TMP) и Text(TMP), настроим по желанию (размеры, шрифт, содержание, расположение и т.д.). Перетаскиваем эти объекты через инспектор в соответствующие поля нашего скрипта (Output Text, Input Field, Generate Button). Запускаем сцену и проверяем работу.

Билд проекта

После успешной проверки работы скрипта переходим в окно Build Settings... (File > Build Settings...), выставляем настройки нашего будущего билда и билдим проект.

Итоги

По итогу был создан проект, который интегрирует языковую модель в Unity и позволяет локально взаимодействовать с ней прямо в игровом движке через UI интерфейс. Это может упростить и ускорить разработку видеоигр, используя языковые модели, например для генерации реплик NPC или даже игровых заданий (квестов). Технология не идеальна, остаётся вопрос по использованию больших языковых моделей, их оптимизации и контроль качества, но это может внести огромный вклад в индустрию.

ПРИЛОЖЕНИЕ А

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.Sentis;
using System.IO;
using System.Text;
using FF = Unity.Sentis.Functional;
using TMPro;
using UnityEngine.UI;

public class RunTinyStories : MonoBehaviour
{
    const BackendType backend = BackendType.GPUCompute;

    string outputString = "";
    const int maxTokens = 100;
    const float predictability = 5f;
    const int END_OF_TEXT = 50256;

    string[] tokens;
    IWorker engine;

    int currentToken = 0;
    int[] outputTokens = new int[maxTokens];

    bool runInference = false;
    const int stopAfter = 100;
    int totalTokens = 0;
```

```
    string[] merges;
    Dictionary<string, int> vocab;

    public TMP_Text outputText;
    public TMP_InputField inputField;
    public Button generateButton;

    int[] whiteSpaceCharacters = new int[256];
    int[] encodedCharacters = new int[256];

    void Start()
    {
        generateButton.onClick.AddListener(OnGenerateButtonClicked);
        SetupWhiteSpaceShifts();
        LoadVocabulary();
    }

    void InitializeModel()
    {
        engine?.Dispose(); // Óäàëÿåì ïðåäûäóùèé äâèæîê, åñëè îí ñóùåñòâóåò
        var model1 = ModelLoader.Load(Path.Join(Application.streamingAssetsPath,
"tinystories.sentis"));
        var model2 = FF.Compile(
            (input, currentToken) =>
            {
                var row = FF.Select(model1.Forward(input)[8], 1, currentToken);
                return FF.Multinomial(predictability * row, 1);
            },
```

```
        (model1.inputs[0], InputDef.Int(new TensorShape()))
    );

    engine = WorkerFactory.CreateWorker(backend, model2);
}

void OnGenerateButtonClicked()
{
    ResetGenerationState();  // Ïîëíîñòüþ ñáðàñûâàåì ñîñòîÿíèå ïðåä çàïóñêîì íîâîé ãåíåðàöèè
    InitializeModel();       // Ñîçäà¸ì íîâûé ýêçåìïëÿð ìîäåëè
    outputString = inputField.text;
    DecodePrompt(outputString);
    runInference = true;
}

void Update()
{
    if (runInference)
    {
        RunInference();
    }
}

void RunInference()
{
    using var tokensSoFar = new TensorInt(new TensorShape(1, maxTokens), outputTokens);
    using var index = new TensorInt(currentToken);
```

```
        engine.Execute(new Dictionary<string, Tensor> { { "input_0", tokensSoFar },
{ "input_1", index } });

        var probs = engine.PeekOutput() as TensorInt;
        probs.CompleteOperationsAndDownload();

        int ID = probs[0];

        if (currentToken >= maxTokens - 1)
        {
            for (int i = 0; i < maxTokens - 1; i++) outputTokens[i] = outputTokens[i +
1];
            currentToken--;
        }

        outputTokens[++currentToken] = ID;
        totalTokens++;

        if (ID == END_OF_TEXT || totalTokens >= stopAfter)
        {
            runInference = false;
        }
        else
        {
            outputString += GetUnicodeText(tokens[ID]);
        }

        outputText.text = outputString;
```

```
    }

    void DecodePrompt(string text)
    {
        var inputTokens = GetTokens(text);

        for (int i = 0; i < inputTokens.Count; i++)
        {
            outputTokens[i] = inputTokens[i];
        }
        currentToken = inputTokens.Count - 1;
    }

    void LoadVocabulary()
    {
        var jsonText = File.ReadAllText(Path.Join(Application.streamingAssetsPath,
"vocab.json"));
        vocab = Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string,
int>>(jsonText);
        tokens = new string[vocab.Count];
        foreach (var item in vocab)
        {
            tokens[item.Value] = item.Key;
        }

        merges = File.ReadAllLines(Path.Join(Application.streamingAssetsPath,
"merges.txt"));
    }

    string GetUnicodeText(string text)
```

```
    {
        var bytes = Encoding.GetEncoding("ISO-8859-
1").GetBytes(ShiftCharacterDown(text));
        return Encoding.UTF8.GetString(bytes);
    }

    string GetASCIIText(string newText)
    {
        var bytes = Encoding.UTF8.GetBytes(newText);
        return ShiftCharacterUp(Encoding.GetEncoding("ISO-8859-
1").GetString(bytes));
    }

    string ShiftCharacterDown(string text)
    {
        string outText = "";
        foreach (char letter in text)
        {
            outText += ((int)letter <= 256) ? letter :
                (char)whiteSpaceCharacters[(int)(letter - 256)];
        }
        return outText;
    }

    string ShiftCharacterUp(string text)
    {
        string outText = "";
        foreach (char letter in text)
        {
```

```
        outText += (char)encodedCharacters[(int)letter];
    }
    return outText;
}

void SetupWhiteSpaceShifts()
{
    for (int i = 0, n = 0; i < 256; i++)
    {
        encodedCharacters[i] = i;
        if (IsWhiteSpace(i))
        {
            encodedCharacters[i] = n + 256;
            whiteSpaceCharacters[n++] = i;
        }
    }
}

bool IsWhiteSpace(int i)
{
    return i <= 32 || (i >= 127 && i <= 160) || i == 173;
}

List<int> GetTokens(string text)
{
    text = GetASCIIText(text);

    var inputTokens = new List<string>();
    foreach (var letter in text)
```

```
        {
            inputTokens.Add(letter.ToString());
        }

        ApplyMerges(inputTokens);

        var ids = new List<int>();
        foreach (var token in inputTokens)
        {
            if (vocab.TryGetValue(token, out int id))
            {
                ids.Add(id);
            }
        }

        return ids;
    }

    void ApplyMerges(List<string> inputTokens)
    {
        foreach (var merge in merges)
        {
            string[] pair = merge.Split(' ');
            int n = 0;
            while (n >= 0)
            {
                n = inputTokens.IndexOf(pair[0], n);
                if (n != -1 && n < inputTokens.Count - 1 && inputTokens[n + 1] ==
pair[1])
```

```
            {
                inputTokens[n] += inputTokens[n + 1];
                inputTokens.RemoveAt(n + 1);
            }
            if (n != -1) n++;
        }
    }
}

void ResetGenerationState()
{
    if (engine != null)
    {
        engine.Dispose(); // Óäàëÿåì ñòàðûé äâèæîê
    }

    currentToken = 0;
    totalTokens = 0;
    outputTokens = new int[maxTokens];
    outputString = "";
}

private void OnDestroy()
{
    engine?.Dispose();
}
}
```