EUROPEAN CLOUD
SUMMIT 2024

Unleashing the Power of Microsoft Semantic Kernel in API Communication

DANIEL COSTEA

MICROSOFT MVP

EUROPEAN CLOUD SUMMIT

Microsoft

run events

AURUM

AvePoint

CoreView

dox42

EasyLife 365

resco

veeam

---

adesso business. people. technology.

Allied Global

ASCENT

BCC

DE CIX

devoteam

empowerID

FPT Software

glueck kanja

Jabra GN

kaspersky

LightningTools

nintex

Rencore

ShareGate:

Spot by NetApp

SysCloud

Syskit

WEBCON LOW-CODE, BUT BETTER.

Use ECS Coins for Swag!

Top 3 win an Atari 2600+

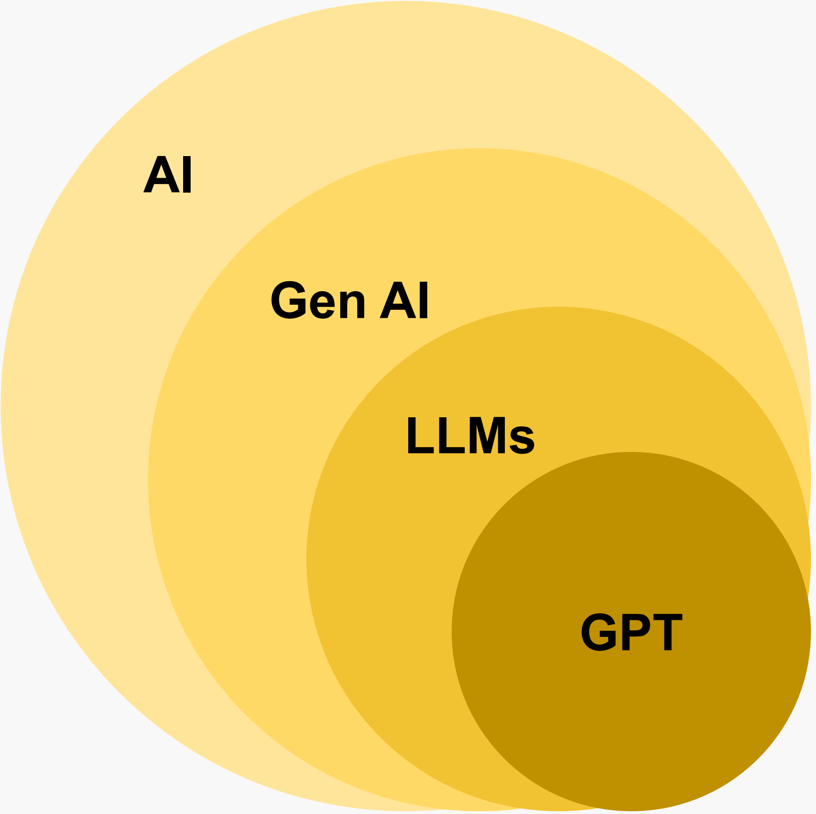
1. Get the app
2. Visit sessions and sponsors, rate sessions
3. Earn ECS Coins
4. Spend ECS Coins

csmmt.eu/app

run.events

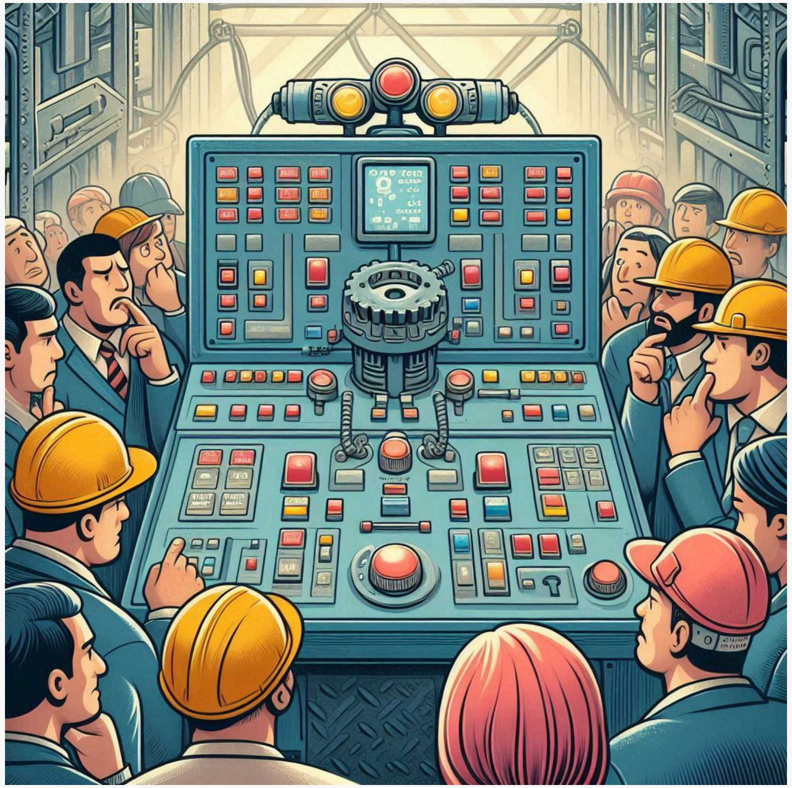
Generative AI / Large Language Models (LLMs)

- Generative AI refers to any AI system whose primary function is to generate content like **text** and **images** in response to input prompts.
- Large Language Models (LLMs) is a subset of Generative AI specialized for the generation and comprehension of human language.
  - LLM is a reasoning engine, not a knowledge store.
  - LLM is a statistical representation of knowledge.
- Generative Pre-trained Transformers (GPT) is a specific subset of LLMs, consisting of a series of models developed by OpenAI



The Problem

- Humans have a problem with machines => UI
- Finally we have a solution => semantic layer
- Coding is not natural!
- The story behind this talk



Semantic Kernel

- Microsoft's Semantic Kernel: An open-source SDK that works with **OpenAI, Azure OpenAI, Hugging Face**. Supported languages: **C#, Python, Java.**
- Advanced Features: Includes **prompt engineering**, AI **orchestrators**, recursive reasoning, and intelligent **planning** for complex problem-solving.
- Access to External Services using **connectors**: Semantic Kernel can leverage vast amounts of information from external services and data.
- Semantic Kernel relies on advanced functionalities, such as **Function Calling** and **OpenAI Assistants** and multimodal capabilities, including vision, image creation (DALL·E 3), and text-to-speech (TTS)

Semantic Kernel Features

- **Plugins** (to organize your native functions and semantic functions in plugins),
- **Function calling** (to let GPT intelligently decide when to call you functions)
- **Planners** (to generate plans to complete even more complex tasks)
- **Personas** (to generate agents, GPT-like instances, able to interact together to solve complex problems).
- **Kernel** (where all these mix together!)

Prompt Templates (Prompt Template Syntax)

- Variables (example):

```
Hello {{$name}}, welcome to Semantic Kernel!
```

- Function calls (example):

```
The weather today is {{weather.getForecast}}.
The weather today is {{weather.getForecast $input}}.
```

- Function parameters (example):

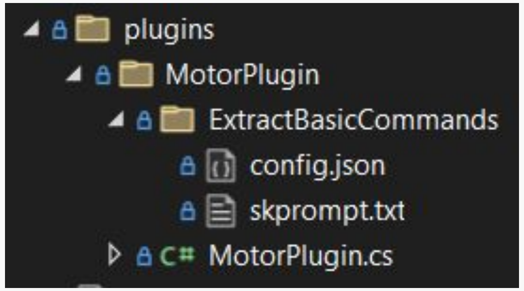
```
The weather today in {{$city}} is {{weather.getForecast $city}}.
The weather today in The Hague is {{weather.getForecast "The Hague"}}.
```

Semantic Functions (using config.json and skprompt.txt)

```
kernel.Plugins.AddFromPromptDirectory("path-to-plugin", "name");
```

skprompt

```
You are a robot car capable of performing only the following
allowed basic commands: {{ $commands }}.
Initial state of the car is stopped.
The last state of the car is stopped.
You need to:
[START ACTION TO BE PERFORMED]
{{ $input }}
[END ACTION TO BE PERFORMED]
Extract a list of basic commands from the action to be performed
to fulfill the goal. Give me the list as a comma separated list.
```



```
plugins
  MotorPlugin
    ExtractBasicCommands
      config.json
      skprompt.txt
    MotorPlugin.cs
```

config

```
{
  "schema": 1,
  "type": "completion",
  "description": "Extract basic motor commands.",
  "completion": {
    "max_tokens": 500, "temperature": 0.0,
    "top_p": 0.0, "presence_penalty": 0.0,
"frequency_penalty": 0.0
  },
  "input_variables": [
    {
      "name": "input",
      "description": "Action to be performed.",
      "is_required": true, "default": ""
    },
    {
      "name": "commands",
      "description": "The commands to choose from.",
      "is_required": true, "default": ""
    }
  ]
}
```

Native Functions

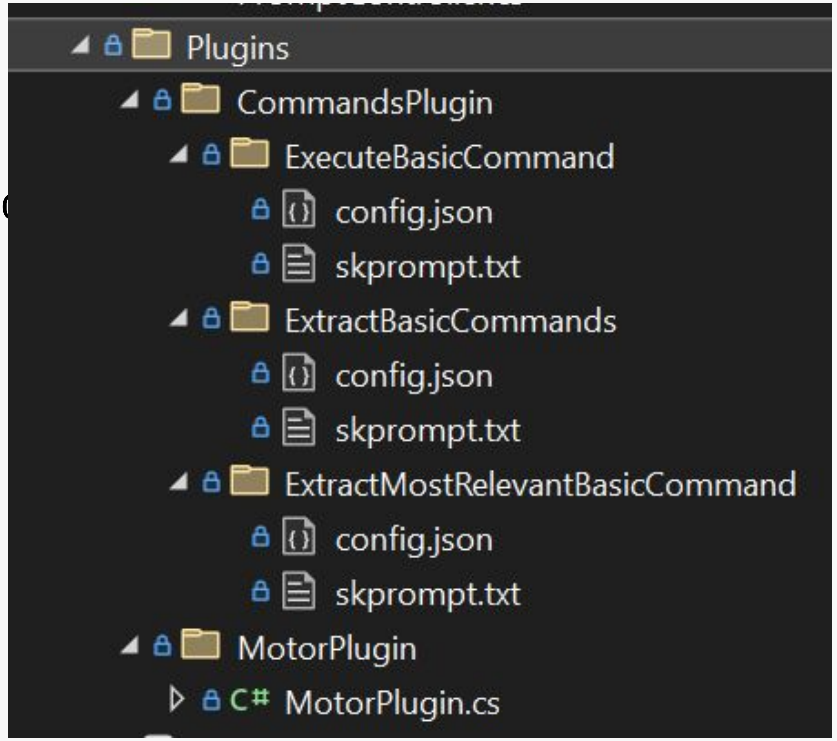
- Native code (C#, Python, Java) annotated with *KernelFunction* attribute
- Perform tasks like: retrieve data, knowing datetime, complex math, memorizing

```
kernel.Plugins.AddFromType<Plugins.MotorPlugin>();
```

```
public sealed class MotorPlugin
{
    [KernelFunction, Description("Moves the car backward.")]
    public void Backward()
    {
        // TODO call car motor API, backward endpoint
    }
    [KernelFunction, Description("Turns the car anticlockwise.")]
    public void TurnLeft()
    {
        // TODO call car motor API, turn left endpoint
    }
}
```

Plugins

- A plugin is a group of **semantic functions** (prompts), **native functions** (traditional code) or a mix between them
- Out-of-the-box plugins available as core plugins:TextPlugin, TimePlugin, MathPlugin, HttpPlugin
- Can be exported so they are usable in ChatGPT, Bing Chat and Micr 365
- More user developed plugins can be added to the kernel



```
Plugins
  CommandsPlugin
    ExecuteBasicCommand
      config.json
      skprompt.txt
    ExtractBasicCommands
      config.json
      skprompt.txt
    ExtractMostRelevantBasicCommand
      config.json
      skprompt.txt
  MotorPlugin
    MotorPlugin.cs
```

FUNCTION CALLING

Function Calling

- **A native feature of LLMs** to generate a JSON payload that includes everything you need to call a function based on the user's intent and the registered functions
- How to work with Function Calling:

1. **Register the functions** (native and semantic functions) to call with the prompt with the kernel.
2. **Sends the prompt:** Prepare and sends the prompt to identify the user's intent.
3. **Function Identification:** LLM returns a JSON payload and identifies the functions that need to be called based on the user's intent (model won't call the functions on your behalf!).
4. **Call the functions:** Matches the identified functions to call in the payload, handle any necessary computations or data retrieval, and send back a response as part of a new message.
5. **Generate final response:** The model processes the new messages which includes the function response, and generates a final response.

Function Calling with Autoinvoke

- ReAct, means that AI is going to call a function, evaluate the response and then call another function if needed
- With *Autoinvoke* (max 5 calls limit!) the function calling gets easier to implement

```
builder.Plugins.AddFromType<Plugins.MotorPlugin>();
…

var settings = new OpenAIPromptExecutionSettings
{
  ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions
};

var streamingResult = kernel.InvokePromptStreamingAsync(ask, new KernelArguments(settings));
await foreach (var streamingResponse in streamingResult)
{
  Log.Debug("STREAMING: {streamingResponse}", streamingResponse);
}
```

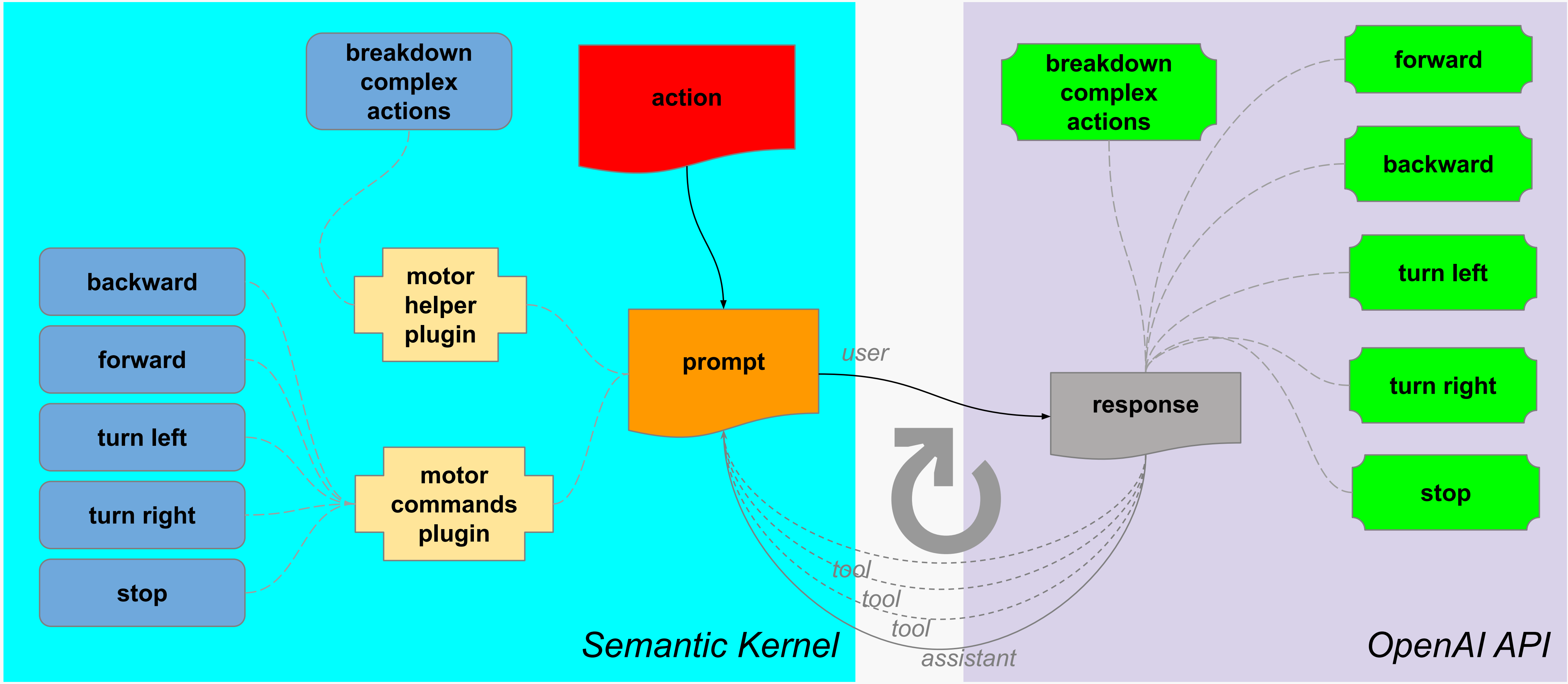
Function Calling

```
builder.Plugins.AddFromType<Plugins.MotorPlugin>();
…
var chatHistory = new ChatHistory();
chatHistory.AddMessage(AuthorRole.User, ask);

var settings = new OpenAIPromptExecutionSettings
{
  ToolCallBehavior = ToolCallBehavior.EnableKernelFunctions,
};
var chatCompletionService = kernel.GetRequiredService<IChatCompletionService>();
var result = await chatCompletionService.GetChatMessageContentAsync(chatHistory, settings, kernel);
var functionCalls = ((OpenAIChatMessageContent)result).GetOpenAIFunctionToolCalls();

foreach (var functionCall in functionCalls)
{
  kernel.Plugins.TryGetFunctionAndArguments(functionCall, out var pluginFunction, out var arguments);
  var functionResult = await clonedKernel.InvokeAsync(pluginFunction!, arguments!);
  var jsonResponse = functionResult.GetValue<object>();
  var json = JsonSerializer.Serialize(jsonResponse);
  chatHistory.AddMessage(AuthorRole.Tool, json);
}
result = await chatCompletionService.GetChatMessageContentAsync(chatHistory, settings, kernel);
```

Function calling



PLANNERS

Planners

- Orchestrator: mix-and-match the plugins/functions registered in the kernel into a series of steps that complete a goal (ReAct = Reasoning Actions)
- Out-of-the-box planners provided by Semantic Kernel include *FunctionCallingStepwisePlanner* and *HandlebarsPlanner*

FunctionCallingStepwise Planner

- An interactive planner based on LLM function calling
- An improved way of identifying the functions to call using AI reasoning
- **FunctionCallingStepwise Planner** works as follows:
    1. **Function Identification:** It identifies the functions that need to be called based on the user’s intent.
    2. **Stepwise Execution:** It executes these functions in a stepwise manner. This means it performs one function at a time, evaluates the result, and then decides on the next function to call.
    3. **Dynamic Evaluation:** It dynamically evaluates the available pathways based on the available assets.
    4. **User Goal Fulfillment:** Its ultimate goal is to fulfill a user’s request or answer a user’s question.

Handlebars Planner

- **Handlebars Planner** generates the plan ahead of the execution in one single LLM call using handlebars template language (not interactive plan!)
- Handlebars prompts work well with iterations (each) and conditions (if-else) decisions using its special syntax
    - Sandbox: https://planetcalc.com/9607/

```
{{!-- Print commands --}}

Permitted commands:
{{#each commands}}
  -  {{this}},
{{/each}}
```

```
{
  "commands": [
    "Forward",
    "Backward",
    "Turn left",
    "Turn right",
    "Stop"
  ]
}
```

```
Permitted commands:
  -  Forward,
  -  Backward,
  -  Turn left,
  -  Turn right,
  -  Stop,
```

Handlebars Planner

- *AllowLoops* allows loops in handlebars planners (not recommended with older GPT 3.5 models)

```
var handlebarsPlannerOptions = new HandlebarsPlannerOptions { AllowLoops = true };

var planner = new HandlebarsPlanner(handlebarsPlannerOptions);

var plan = await planner.CreatePlanAsync(kernel, ask);

Console.WriteLine("GENERATED PLAN: {prompt}", plan.Prompt);

result = await plan.InvokeAsync(kernel, variables);

Console.WriteLine("RESULT: {result}", result.Trim());
```

Robot Car Powered by Semantic Kernel

Direction	Arrow
Forward	→
Backward	←
Turn left	↗
Turn right	↘
Stop	▪

Example: left, right, forward, stop: ↗ ↘ → ·

DEMO

Conclusions

	Function calling w/ auto invoke	Function calling	Function calling stepwise planner	Handlebars planner
Complexity	easy	complex	easy	easy
Speed	fast	fast	slower	slow
Cost	cheap	cheap	expensive	expensive
Features	max 5 tool calls supports streaming	unlimited tool calls supports streaming	dynamic selection of tools user-defined calls limit user-defined calls interval	supports handlebars syntax review the plan before execution save the plan supports loops and statements

Resources

Semantic Kernel
https://learn.microsoft.com/en-us/semantic-kernel
https://github.com/MicrosoftDocs/semantic-kernel-docs
https://devblogs.microsoft.com/semantic-kernel
https://github.com/microsoft/semantic-kernel

Prompt engineering
https://www.promptingguide.ai

OpenAI
https://learn.microsoft.com/en-us/azure/ai-services/openai/reference
https://platform.openai.com/docs/api-reference

Demo repository
https://github.com/dcostea/Apex.RobotCarGpt

THANK YOU,
YOU ARE AWESOME ❤️

PLEASE RATE THIS SESSION IN THE MOBILE APP.

LinkedIn	https://linkedin.com/in/danielcostea
Twitter	https://twitter.com/dfcostea
Blog	http://apexcode.ro
Email	daniel_costea@ymail.com
GitHub	https://github.com/dcostea

MVP Microsoft® Most Valuable Professional