Style Definition: Cover Title,ct

Transparency Note: GPTools for GenAI Scripting

Updated February 6, 2024

Table of Contents

The basics of GPTools

Introduction

GPTools is a framework that empowers teams, including non-developers, to create and use AI-enhanced scripts to support their workflows. GPTools provides support for authoring and debugging JavaScript scripts that incorporate calls to foundation models and LLMs in their execution[1]. GPTools is a programming framework that allows its users to author AI scripts (which we call a GPTool), debug those scripts in a development environment that is an extension of VS Code, and package those scripts in a command-line interface that can be deployed in many contexts.

Our VS Code extension supports easy authoring of a GPTools by writing natural language in markdown syntax plus a small amount of stylized JavaScript programming. Our framework allows users to leverage multiple LLM models, parameterize the calls to the models, execute and debug scripts, trace the construction of the LLM prompts and provide a full trace of execution from prompt construction to LLM generation to parsing the LLM result. Our framework also supports extracting multiple forms of output from LLM generations, including output in files of different types, outputs intended as edits to existing files and outputs in structured formats, such as JSON.

Key terms

GPTool A stylized JavaScript program that defines the context for the LLM call, allows arbitrary JavaScript code execution, packages the prompt input for the LLM, calls the LLM, and unpacks that LLM output based on the directions given in the prompt.

GPSpec An optional markdown file(s) used to establish the calling context for the GPTool. A GPSpec can be as simple as a list of arguments and as complex as a collection of markdown documents in a repository.

GPVM: A runtime system that given a GPTool and an optional GPSpec, executes the GPTool, which involves integrating the context into a prompt, calling the specified LLM, and extracting content from the LLM result.

VS Code GPTools extension An add-in to VS Code that provides users with easy methods for creating, editing, running and debugging GPTools.

Foundation models and LLMs While GPTools currently supports different LLMs, in the future we anticipate that we will incorporate additional foundation models beyond large language models.

Capabilities

System behavior

GPTools is a general-purpose AI-script authoring framework for seamlessly integrating code execution and foundation model/LLM invocations. A GPTool is a JavaScript program in a stylized format that allows users to easily specify the LLM context and prompt, invoked a specified model, and parse the resulting output according to user specifications. This functionality allows even users who are not programmers to inspect model results and double check them for correctness.

[1] Throughout this document when we refer to LLMs we mean any foundation model that is compatible with our interfaces.

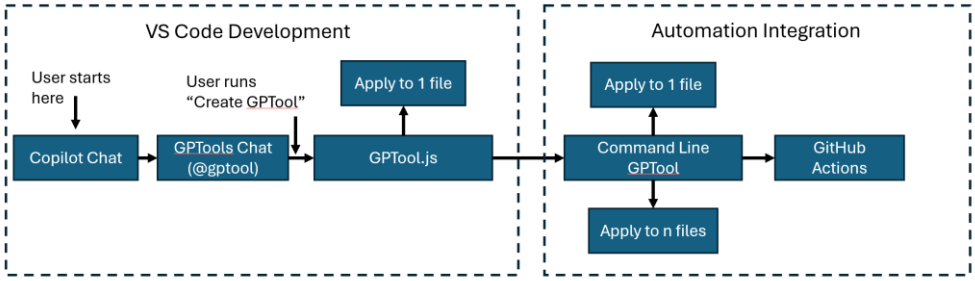
GPTools can be written in any IDE but the VS Code GPTools add-in makes creating, executing and debugging GPTools especially easy. GPTools users can implement tools that generate and edit multiple files with a single tool and our integration with VS Code leverages existing functionality in for refactoring to allow users to easily see the results of the tool execution. The add-in supports creating a new GPTool, creating a new GPSpec, invoking a given GPTool from a GPSpec, tracing the execution of the GPTool in establishing the LLM context and final prompt, and unparsing the LLM output into the user-specified elements. Examples of all of these capabilities can be viewed in demo videos in the GPTools repository: [microsoft/gptools: GenAI Scripting (github.com)]

The goal of GPTools is to empower a broad range of potential users to innovate with building AI-powered scripts and identify new ways to leverage AI for their daily tasks. We expect that professional developers, who are familiar with writing and using scripts to enhance their productivity will be the early adopters of GPTools. GPTools will give these users benefit because GPTools can do many things that existing scripts written in traditional scripting languages like JavaScript and Python cannot do. While developers can leverage other frameworks, such as langchain and Semantic Kernel, that integrate calls to LLMs into languages like Python and JavaScript, these frameworks require more user effort and have less IDE support than GPTools. Ultimately, because our goal is to make GPTools easy to author, modify, debug and run, we anticipate that they will be useful far beyond professional developers. The real impact of GPTools will be to enable many non-developers to innovate and build GPTools that enhance their productivity. We illustrate this point with examples below.

Documentation

To help users get started with GPTools, we include documentation in our repository that illustrates in code snippets the contents of several different GPTools. The documentation shows both what the example GPTool looks like as well as what the effect is from the GPTool acting on a particular input. While these examples are intended to explain the technology, they are not intended to be the basis for user-written tools.

GPTools User Journey



Our goal is to provide a seamless transition from using a chat experience in an IDE such as VS Code to authoring, debugging, and deploying GPTools. We illustrate the user journey in the figure above. The user starts by interacting with the existing Copilot Chat capability inside VS Code. If they want to interact with GPTools using VS Code chat, they can invoke the @gptools add-in directly in the VS Code chat. When they do this, they will implicitly start building a GPTool, leveraging the interfaces that exist in GPTools for all users. At any point, the user can explicitly request that a GPTool be created from their chat history and our framework will create a new GPTool file that they can further edit, run, and debug. This interactive experience is illustrated on the left side of the figure. At some point, a user will have confidence that the GPTool they have created is good enough that they want to use it at scale (perhaps applying it to all the files in a directory or repository). To facilitate this, users can create a command line instance of a GPTool that can then be integrated into other scripting and automation mechanisms, such as running the tool over all the files in a directory or integrating the tool as GitHub Action.

Commented [BZ1]: @Alicia Edelman Pelton (SHE/HER) - this is new content.

Use cases

Intended uses

GPTools can be used in any context where a command line script written in another programming language might be used but the use cases are much more ambitious because the LLM can do much more than ordinary code. Here are some examples:

- **Checking for potential inconsistencies in a collection of configuration files or other content.** Using the LLM, a GPTool can inspect configuration files and leverage the LLM's understanding of common configuration errors to detect and report them. Before LLMs, professional developers would write tools, such as lint[2], which are complex programs that detect inconsistencies in the syntax of their code files. With GPTools, checking tools can be written for much richer scenarios (such as checking for inappropriate variable names), and by individuals who are not professional developers.
- **Automating document translation:** Given documentation in a repository written in one natural language, a GPTool can be written to translate that documentation into another language. For a specific example of why GPTools is important for this use, consider the task of maintaining the localization of the MakeCode[3] documentation. MakeCode documentation has nearly 2M files, which are typically markdown with a mix of code snippets. Many documents are partially translated (paragraph level). There are 3500 registered volunteer translators for 35+ languages. One cannot just apply Bing translate as it typically destroys the code snippets. With GPTools, we can have a script that goes through every documentation file, pulls the current localized version and assembles a prompt to ask the LLM to fill in the missing translations, while leaving the existing ones alone. The LLM already crawled MakeCode and is aware of the syntax.
- **Generating executable code from a natural language specification.** A GPSpec file can be used to specify the task being performed and a GPTool that specializes in code generation can translate the spec into code.
- **Creating a short version of a longer white paper by summarizing each chapter.** LLMs are quite effective at summarizing documents. A GPTool can be written to take each chapter of a long document and summarize it in a section of a shorter document.

Commented [BZ2]: @Alicia Edelman Pelton (SHE/HER) - rewrote this part

[2] Lint (software) - Wikipedia
[3] https://makecode.org/

- **Translating a monolog to a dialog.** Given a monolog from a video transcript, a GPTool can be written to rewrite the monolog into a dialog between two individuals (akin to sports announcers talking to each other) to make the video more interesting and accessible.

Unintended uses

GPTools is a general framework for authoring scripts. As a result, an adversary can use GPTools to author adversarial scripts that could be used for malicious purposes. All of the adversarial uses of GPTools could also be implemented in other LLM language extension frameworks such as Sematic Kernel, autogen, and langchain, so the danger from unintended uses of GPTools stems from possibility that it might make it easier to author adversarial scripts. This issue is present in any infrastructure that makes programming easier, including languages such as PowerShell, JavaScript, and Python, as well as IDEs such as VS Code and Visual Studio. While we cannot prevent unintended uses, we will encourage users to consider Responsible AI practices when then build GPTools.

Foundation model best practices

We strongly encourage GPTools users to use foundation models and LLMs that support robust Responsible AI mitigations, such as the Azure Open AI (AOAI) services. Such services continually update the safety and RAI mitigations to track our up-to-date understanding on how to deploy and use foundation models most responsibly. This blog post highlights the RAI features in AOAI that were presented at Ignite 2023:

Announcing new AI Safety & Responsible AI features in Azure OpenAI Service at Ignite 2023 (microsoft.com)

Commented [BZ3]: @Alicia Edelman Pelton (SHE/HER) - this is new content

Limitations

GPTools is an evolving framework that will improve based on input from users. Existing limitations in the framework include weak support for Retrieval Augmented Generation (RAG), integration into only one IDE (VS code), and internal support for OpenAI APIs plus a relatively small number of other LLMs. We intend to allow users to integrate calls to external services (such as RAG) in GPTools to provide the LLM with more context. In these cases, users can choose to register a set of pre-existing functions and ask the LLM to choose which function to call and with what arguments. Such uses of LLMs have risks related to adversarial and poorly written extensions, which the user has the ability to choose and control. Our documentation will highlight the potential risks of such use cases, and recommend caution when using this feature.

Commented [BZ4]: @Alicia Edelman Pelton (SHE/HER) - added this comment about a user building a GPTool that calls other services.

We anticipate adding support for more foundation models as the use cases evolve.

We also anticipate that the on-ramp to using GPTools will evolve. We currently support invoking the GPTools framework as part of a VS Code Copilot Chat experience (hosted in VS Code Insider's Edition). We are exploring ways to generate a GPTool directly from the user chat interactions which will lower the barrier to entry for using GPTools. We also understand that some developers would prefer to implement their GPTools using Python instead of JavaScript. We anticipate building a Python binding form authoring GPTools in the future.

Technical limitations, operational factors and ranges

GPTools does not use any AI model in executing the framework itself. Individuals using GPTools to author their own AI scripts will be subject to the technical limitations, operational factors, and ranges of the AI LLM their tool uses.

System performance and errors

Not applicable.

Best practices for improving system performance

GPTools encourages users to consult the best practices for authoring effective prompts for the specific LLM they are invoking in their tool.

Evaluation of the GPTools

Not applicable.

Learn more about responsible AI

Microsoft AI principles

Microsoft responsible AI resources

Microsoft Azure Learning courses on responsible AI

Learn more about the GPTools

Read more about GTools at our GitHub site, microsoft/gptools: GenAI Scripting (github.com)

Contact us

Give us feedback on this document: zorn@microsoft.com, jhalleux@microsoft.com

About this document

This document is not intended to be, and should not be construed as providing. legal advice. The jurisdiction in which you're operating may have various regulatory or legal requirements that apply to your AI system. Consult a legal specialist if you are uncertain about laws or regulations that might apply to your system, especially if you think those might impact these recommendations. Be aware that not all of these recommendations and resources will be appropriate for every scenario, and conversely, these recommendations and resources may be insufficient for some scenarios.

Published: <date>

Last updated: <date>