GenAIScript: Robust Scripting for Generative AI

Peli de Halleux, Markus Kuppe, Michal Moskal, Madan Musuvathi, Benjamin Zorn

April 2024

Abstract

Scripting languages, like Python and JavaScript, have enabled major advances in computing but were created prior to the advent of Large Language Models (LLMs). We introduce GenAIScript, a scripting language that makes LLMs a first-class part of the scripting process, allowing users to author, debug, and deploy LLM-based scripts that can perform tasks beyond the reach of conventional code. GenAIScript is intended to be used by users with different programming experience and applied to diverse tasks previously unable to be solved using traditional code. By extending the popular Visual Studio Code environment, GenAIScript simplifies the creation of robust and reusable scripts that can intelligently process and generate content, including code, documentation, and structured data. We explain what GenAIScript is, how it is implemented, and illustrate its use in several production deployments. GenAIScript is open-source and available to use now at: https://aka.ms/genaiscript.

1. Introduction

GenAIScript is a JavaScript-compatible[1] scripting language that allows users to write scripts to automate complex tasks by leveraging LLMs as a first-class script element.

The dramatic rise in the capabilities of artificial intelligence has revolutionized the way we interact with technology, offering unprecedented opportunities to automate and enhance many tasks. Incorporating calls to LLMs (or more generally any foundation model) at runtime into a software system greatly enhances what that software can do. The disruptive nature of this transition forces us to categorize software into Plain Ordinary Software (POSW), which does not use the power of LLMs at runtime, and the more capable AI Software (AISW) that does leverage LLMs[2].

POSW represents all software written before the creation of LLMs and includes operating systems, applications, scripts, etc. While POSW is a key element of our existing computing infrastructure, AISW, with its enhanced capabilities, will dominate the development of future software systems. Our goal is to make GenAIScript the language for scripting AISW.

Historically, scripting languages like Perl, JavaScript, and Python, have become widely successful because they provide two key elements: (1) they allow diverse users to write small but useful programs, and (2) they give users access to new computing capabilities not previously available to programs (e.g., JavaScript gave its users access to the Web).

Given that context, the goal of GenAIScript is to empower a wide class of potential users to easily leverage the power of AISW. GenAIScript enables users, including those without extensive programming expertise, to author, debug, and deploy scripts that seamlessly incorporate calls to LLMs. GenAIScript is a JavaScript-compatible language with a dedicated VS Code extension to simplify the scripting process, making it accessible to a broad audience.

What are GenAIScripts used for? We believe that the incredible flexibility of LLMs allows our scripts to be used in many contexts for many purposes. Some of the scripts we have personally explored include: extracting information from images using the gpt-4-turbo-v model, extracting exact quotes from documents to determine if the content answers specific review questions, planning a vacation to a destination specified as a parameter leveraging information from search and web-hosted documents about the

[1] Currently GenAIScripts are JavaScript-compatible but in the future we anticipate bindings for other languages including Python.

[2] For a discussion of the implications of POSW versus AISW, see the SIGPLAN blog post: AI Software Should be More Like Plain Old Software | SIGPLAN Blog

destination, generating critiques of documents with respect to tone, content, and audience, and performing systematic natural language translation of text embedded in software artifacts (see the case study).

With the goal of making GenAIScript the go-to language for writing AISW, we have the following key objectives:

Commented [BZ1]: @Madan Musuvathi - I added a list of key objectives and how we address them related to your question. Let me know what you think.

1. **Integration with LLMs**: GenAIScript is designed to make LLMs a first-class part of the scripting process, allowing users to author, debug, and deploy LLM-based scripts.

2. **Simple Abstractions**: GenAIScript provides abstractions to define the prompt context, refer to the context in the prompt itself, and supports specification and checking of multiple output formats.

3. **Ease of Use**: GenAIScript targets a wide class of potential users, including non-developers. The VS Code extension and integration with existing IDE features contribute to this quality.

4. **Robustness**: GenAIScript supports building robust scripts by leveraging the capabilities of VS Code for writing, executing, debugging, and refining scripts. Additional features like input and output specifications and static orchestration graphs also contribute to robustness.

5. **Seamless Integration with Plain Old Software (POSW)**: GenAIScript is designed to be embedded into existing POSW workflows, allowing users to integrate AI capabilities into traditional software systems.

The contributions of this paper are:

- The design and implementation of the first scripting language where the LLM is a first-class element.
- The design of the GenAIScript runtime that supports parameterized and structured input of LLM context and supports constraints and systematic verification of the LLM and script outputs.
- Case studies of the application of GenAIScript in two diverse deployment contexts.

In the remainder of this paper, we describe how to write a GenAIScript, define key GenAIScript abstractions, highlight the capabilities of the framework, and describe several case studies based on our experiences building and deploying GenAIScripts in production setting. We discuss related work and our thoughts about future additions and features.

2. Introducing GenAIScript

GenAIScript was designed to provide a low barrier to entry so that many users, including non-developers, can quickly start using and writing scripts. However, we also want to enable power-users to leverage their programing skills to create and deploy more sophisticated scripts for more challenging scenarios. As a result, GeneAIScript is JavaScript compatible where the scripts can contain JavaScript code but less sophisticated users can still define powerful and effective scripts using the simple abstractions we provide.

To understand GenAIScript, we present a simple complete script in Figure 1.

```
// metadata and model configuration
script({ title: "Shorten", model: "gpt4" })
// insert the context
def("FILE", env.files)
// appends text to the prompt
$`Shorten the following FILE. Limit changes to minimum.`
```

Figure 1: A simple GenAIScript

This example illustrates the key elements of every GenAIScript: the metadata, the context, and the prompt. We describe each element in turn.

Commented [BZ2]: @Peli de Halleux , @Markus Kuppe , @Madan Musuvathi I added this description to the white paper. Let me know what you think.

Because every GenAIScript calls an LLM, metadata is needed to define the parameters that will be used to invoke the LLM. Those parameters can include the model, the temperature, the number of tokens to return, etc. By default, GenAIScripts use the “gpt-4” model and other reasonable defaults, so many of the metadata fields can be omitted.

A GenAIScript executes in three parts: 1) set up the LLM prompt, 2) execute the LLM, and 3) parse the LLM output.

Setting up the prompt is done with the “def” command, which in this case associates the name FILES with the contents of env.files. env.files is a variable that refers to the parameters (typically one or more files) that the script is invoked on. After this def command, when the user refers to FILES in the prompt, the LLM will know they are referring to the contents of the files in env.files.

The text between $`…` in the figure is the prompt that the LLM will process. The $`…` instructs the LLM what to do and refers to the names defined using def such as FILE. This prompt is typically the final part