Component Augmented Generative Experessions (CAGE)

Design Prompt to Code Interactions

Ryan Morton 2025

Overview

- Define CAGE
- Define workflow
- Language binding workflow
- Task generating workflow
- Feature generating workflow
- Scaffolding generating workflow
- Code Generating Application High-level Design

Component Augmented Generative Expression

A definition

- Component Augmented Generative Expression relates semantic meaning to LLM generated responses in a programmatic way to produce modular, composable code.
- A component may have four levels of abstraction
  - Language bindings: converts language to low-level functions in the programming language.
  - Tasks: performs a series of actions using language bound functions.
  - Features: composes a fully realized application element using task functions.
  - Scaffolding: generates boilerplate application code, layouts, resources, and places feature functions inside that boilerplate code. This is the only element that necessarily combines programming language specific code with abstracted features.

Component Augmented Generative Expressions

A definition

- Components levels may be defined by the variation of semantic meaning to generate:
    - Language bindings should generally converge the semantic meaning of the output even with divergent requests.
    - Task functions should allow divergent workflows with the possibility of convergence for repeated tasks. The LLM should generally produce pseudo-code with occasional language specific code. Ideally, tasks should be easily translated between programming languages.
    - Feature functions enable the most open-ended semantic meaning for requests and responses as the LLM should not have to produce code when they are generated. Ideally, features should be easily translated between programming languages.
    - Scaffolding functions should converge on specific types of functions and layouts with variation of titles, colors, sizing, and other design elements.

Component Augmented Generative Expressions

A definition

- The central thesis of the project:
    - Highly variant semantic meaning should be directed to the design of tasks and features while strictly controlled language bindings ensure predictable behavior.
    - Keep the creativity of the LLM to higher level abstractions to exploit adaptability.
    - Language bindings need only pick a single correct way to perform small actions even if there are many correct ways.
    - Reserve scaffolding for boilerplates that don’t require creative LLM responses.

CAGE Workflow

A definition

- A workflow mixes API software with LLM chat to perform work a programmer would need to perform:
  - Write code
  - Write documentation
  - Write unit tests
  - Design features
- Each workflow has a unique chat to focus attention on that task.
- API software can run workflows in cycles and communicate with each other when needed.

CAGE Workflow

A definition

- Each workflow has a unique and specific system prompt for its task.
- System prompts provide the context, rules, instructions, and examples the LLM needs to generate responses to API software requests.
- Each component type has its own system prompt and workflow.

Language Binding Functions

Tying the LLM to a programming language

- Language bindings provide the base layer upon which greater abstraction may be applied.
- Bindings perform a specific action, validate inputs, validate outputs (if used), and should generally converge around a way of performing that action.
- If a task or feature requires a binding function that is similar to an existing one, API software should request an update to the existing binding function.
- Task functions produce the pseudo-code as a prompt to this workflow.
- Repeatable bindings should be stored as a separate function in the codebase. Others may be generated only for its specific task. API software could make this configurable.

Language Binding Functions

Ensuring quality

- Binding functions should be documented and tested by separate workflows.
- API software should make several code generating attempts to pass unit testing. Attempts may be configurable by the user.
- LLM fine-tuning can be applied over time for request-response pairings the user or code-reviewer approves.

Task Generating Workflow

Performing several actions

- Task generation produces a function where the input and outputs are known but the body is pseudo-code.
- While related tasks may use the same language bindings over and over, the order in which they occur may change.
- Pseduo-code will be used by the language binding workflow in the form of a request.
- Tasks focus on the “what” for an application, not the “why” or the “how”/

Task Generating Workflows

Ensuring quality

- Once a task is fully realized, it should also be documented and tested.
- API software should make several code generating attempts to pass unit testing. Attempts may be configurable by the user.
- LLM fine-tuning can be applied over time for request-response pairings the user or code-reviewer approves.
- Tasks generation should support multiple language bindings with only minor alterations: defining a function, assignment usage, spacing conventions.

Feature Generation Workflows
Supporting Any Idea

- Feature generation workflows define the purposes of the application.
- LLMs should produce fully formed features that focus on the “why” and “how” not the “what” of an application. For example:
    - User Controls create different experiences relative to a domain and datasete.
    - Data Visualizations augment cognition relative to a domain and dataset.
    - An API supports micro-service architecture relative to multiple front-ends, domains, and datasets.