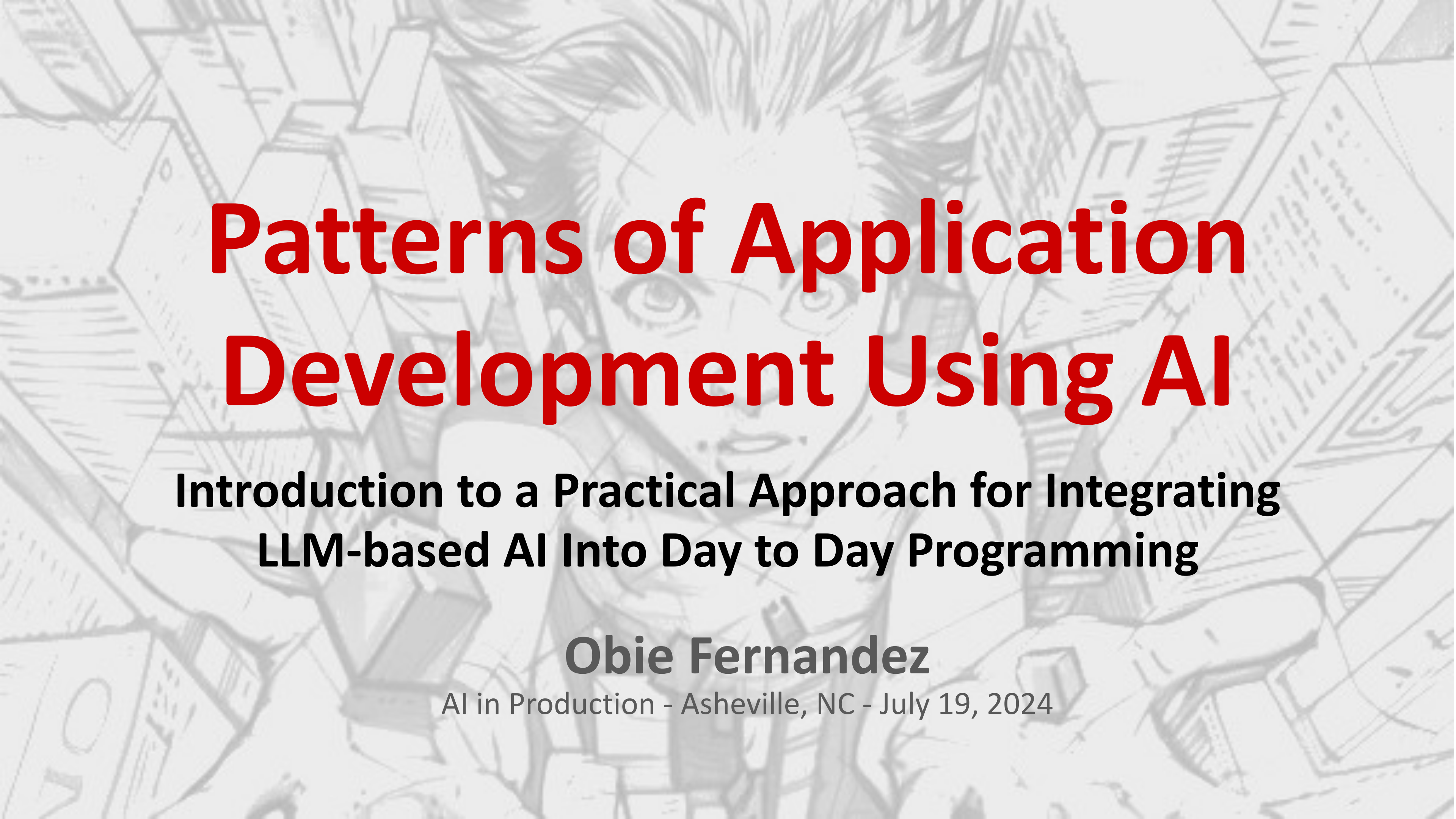


Patterns of Application Development Using AI

Introduction to a Practical Approach for Integrating LLM-based AI Into Day to Day Programming

Obie Fernandez

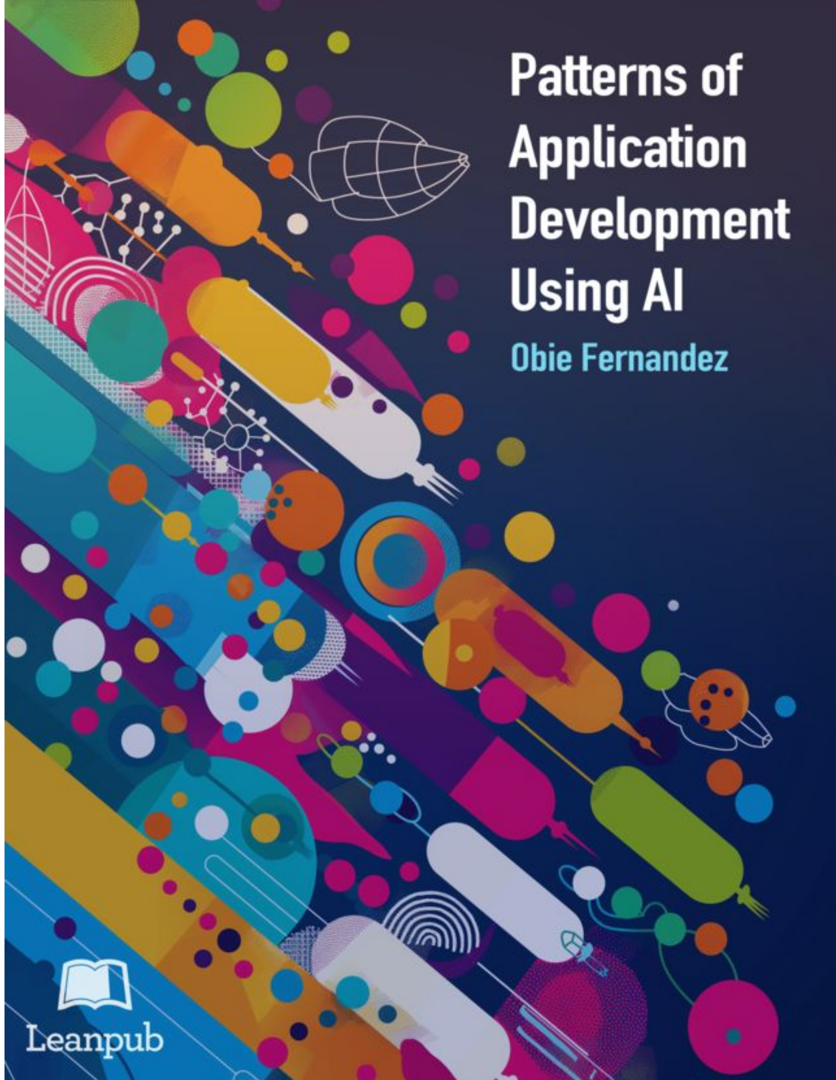
AI in Production - Asheville, NC - July 19, 2024

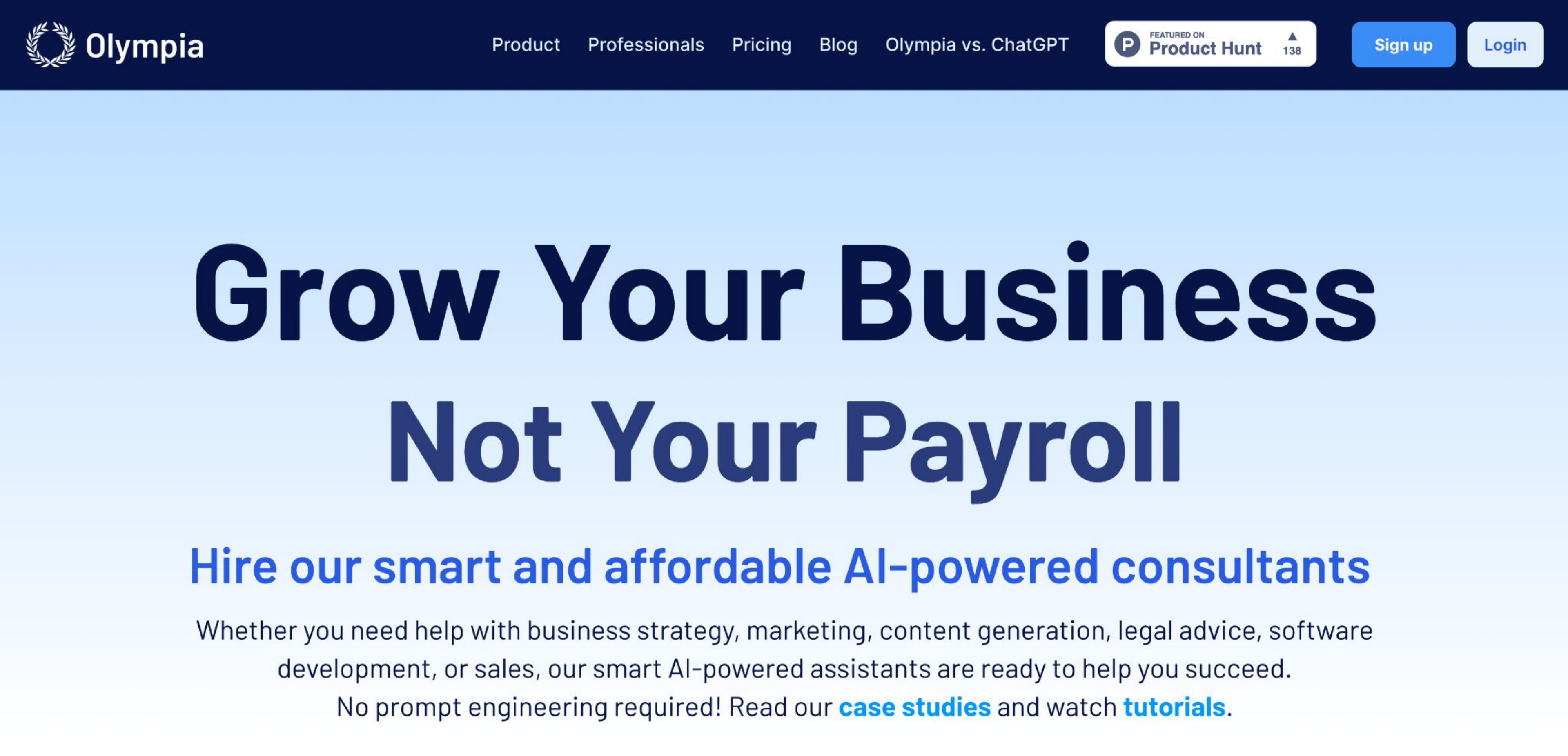
Introduction

Generative AI and LLMs in particular are revolutionizing how I write software. I wrote a book about it.

Key Question: How to apply this approach without *changing everything?*

In this talk I will discuss one abstract pattern and at least a couple of concrete patterns depending on time constraints.





Grow Your Business
Not Your Payroll

Hire our smart and affordable AI-powered consultants

Whether you need help with business strategy, marketing, content generation, legal advice, software development, or sales, our smart AI-powered assistants are ready to help you succeed.
No prompt engineering required! Read our **case studies** and watch **tutorials**.

https://olympia.chat

The Abstract Patterns

Narrow The Path
Focusing the AI on the task at hand so it's not distracted by its vast latent space

Retrieval Augmented Generation (RAG)
Retrieve relevant information then combine with prompt to provide richer context

Multitude of Workers
Decompose workflows into collaborating almost-human discrete components

Tool Use
Functions that an LLM can interact with during the generation process

Self Healing Data
Automatically detect, diagnose, and correct data anomalies, inconsistencies, or errors

Contextual Content Generation
Generate dynamic and context-specific content within your applications

Generative UI
Create highly personalized and dynamic user experiences on-the-fly

Intelligent Workflow Orchestration
Dynamically orchestrate and optimize complex workflows

The Concrete Patterns (28)

Prompt Engineering

Chain Of Thought
Role Assignment
Prompt Object
Prompt Template
Structured IO
Prompt Chaining
Prompt Rewriter
Response Fencing
Query Analyzer
Query Rewriter
Ventriloquist

Discrete Components

Predicate
API Facade
Result Interpreter
Virtual Machine

Human In The Loop

Escalation
Feedback Loop
Passive Information Radiation
Collaborative Decision Making
Continuous Learning

Intelligent Error Handling

Contextual Error Diagnosis
Intelligent Error Reporting
Predictive Error Prevention
Smart Error Recovery
Personalized Error Communication
Adaptive Error Handling Workflow

Quality Control

Eval
Guardrail

Multitude of Workers

Decompose workflows into collaborating almost-human discrete components

I like to think of my AI components as little, almost-human virtual “workers” that can be seamlessly integrated into my application logic to perform specific tasks or make complex decisions.

Discrete AI Components

Distinct AI-powered building blocks that streamline the process of incorporating intelligent behaviors into your software

AI can help you declare parts of your biz logic in plain language.

Stepping stone to writing entire applications in prompt-driven style

Modularity is Key

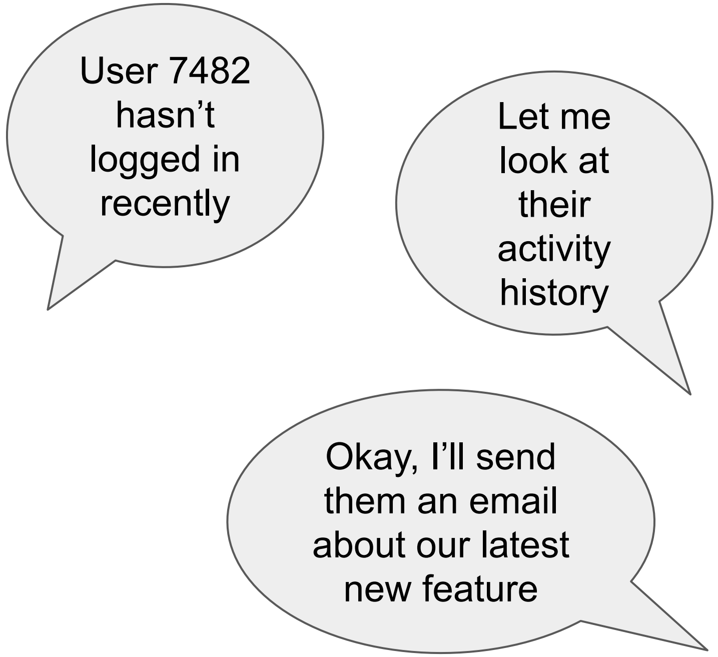
Proponents of OO programming tell us to think about object interactions as messages.

AI workers "talk to each other" via plain language messages, like little humans interacting

Loosely-coupled approach that allows your application to adapt and evolve over time

Influenced by Microservices approach

Different than typical agents approach, these workers do stuff on behalf of your application, not the user


User 7482 hasn't logged in recently
Let me look at their activity history
Okay, I'll send them an email about our latest new feature

Inspired by Behavior Driven Development (BDD)

BDD emphasizes collaboration among developers and non-technical stakeholders to understand a system's expected behavior through clear examples written in an easily understandable language

Literally invented by many of my Rubyist colleagues at Thoughtworks including:

- Dan North
- Liz Keogh
- **Aslak Hellesoy (Cucumber)**
- Dave Astels

```
Given I am on the home page
When I click on the "Login" button
Then I should see the login form
```

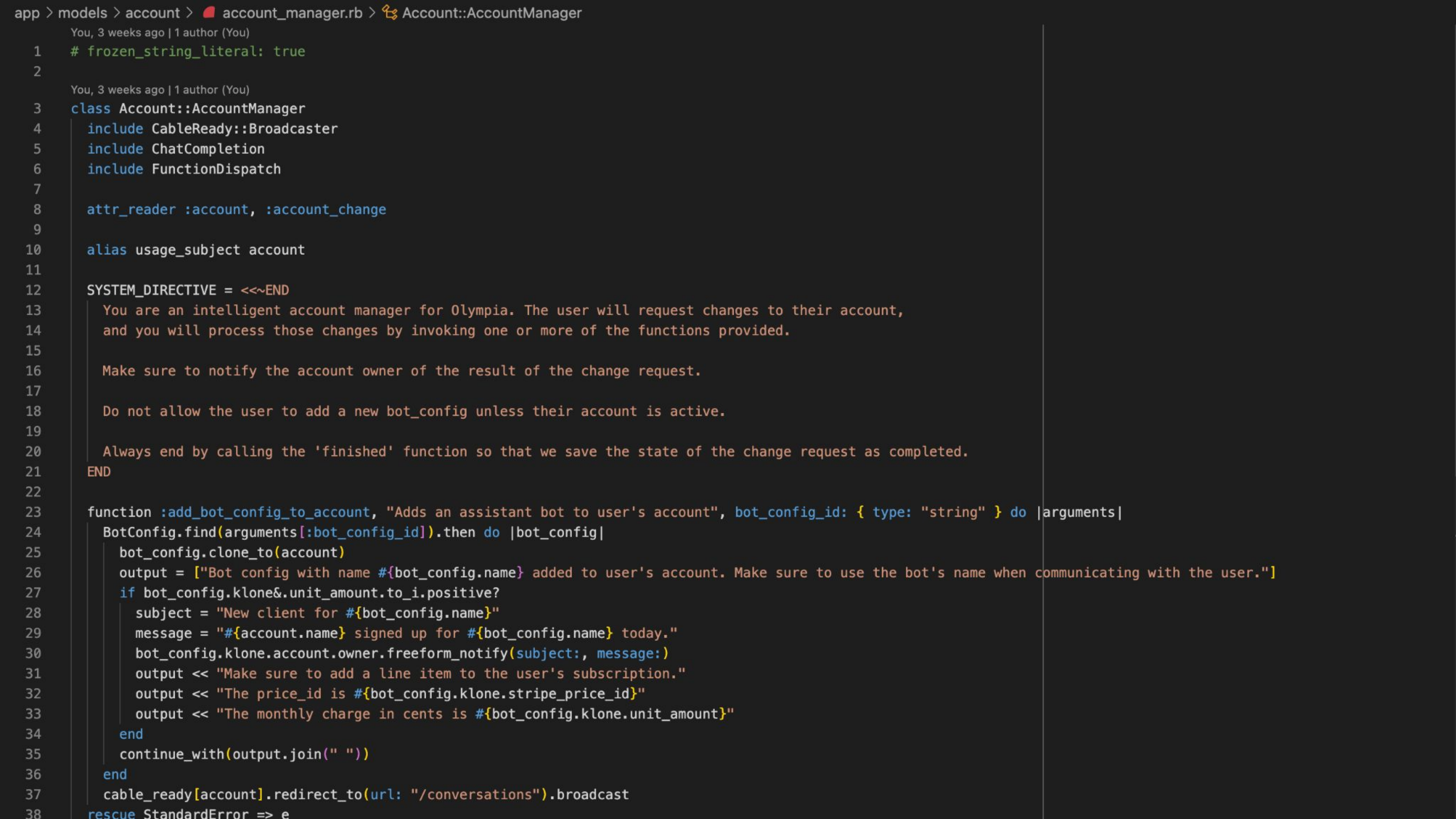
Example: Account Manager

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking one
or more of the functions provided. Escalate to human support rep if you
encounter any problems.

Do not allow the user to change their account or add a new AI assistant
unless their account subscription status is active.

Make sure to notify the account owner of the result of the change request
whether or not it is successful.

Always end by calling the 'finished' function so that we save the state
of the change request as completed.
```


app > models > account > account_manager.rb > Account::AccountManager

```ruby
You, 3 weeks ago | 1 author (You)
# frozen_string_literal: true

You, 3 weeks ago | 1 author (You)
class Account::AccountManager
  include CableReady::Broadcaster
  include ChatCompletion
  include FunctionDispatch

  attr_reader :account, :account_change

  alias usage_subject account

  SYSTEM_DIRECTIVE = <<~END
    You are an intelligent account manager for Olympia. The user will request changes to their account,
    and you will process those changes by invoking one or more of the functions provided.

    Make sure to notify the account owner of the result of the change request.

    Do not allow the user to add a new bot_config unless their account is active.

    Always end by calling the 'finished' function so that we save the state of the change request as completed.
  END

  function :add_bot_config_to_account, "Adds an assistant bot to user's account", bot_config_id: { type: "string" } do |arguments|
    BotConfig.find(arguments[:bot_config_id]).then do |bot_config|
      bot_config.clone_to(account)
      output = ["Bot config with name #{bot_config.name} added to user's account. Make sure to use the bot's name when communicating with the user."]
      if bot_config.klone&.unit_amount.to_i.positive?
        subject = "New client for #{bot_config.name}"
        message = "#{account.name} signed up for #{bot_config.name} today."
        bot_config.klone.account.owner.freeform_notify(subject:, message:)
        output << "Make sure to add a line item to the user's subscription."
        output << "The price_id is #{bot_config.klone.stripe_price_id}"
        output << "The monthly charge in cents is #{bot_config.klone.unit_amount}"
      end
      continue_with(output.join(" "))
    end
    cable_ready[account].redirect_to(url: "/conversations").broadcast
  rescue StandardError => e
```

Business Logic Revisions

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking one
or more of the functions provided. Escalate to human support rep if you
encounter any problems.

Do not allow the user to change their account or add a new AI assistant
unless their account subscription status is active.

Make sure to notify the account owner of the result of the change request
whether or not it is successful.

Always end by calling the 'finished' function so that we save the state
of the change request as completed.
```

Prompt Engineering

Patterns for optimizing prompts, improving AI performance, and achieving desired outcomes

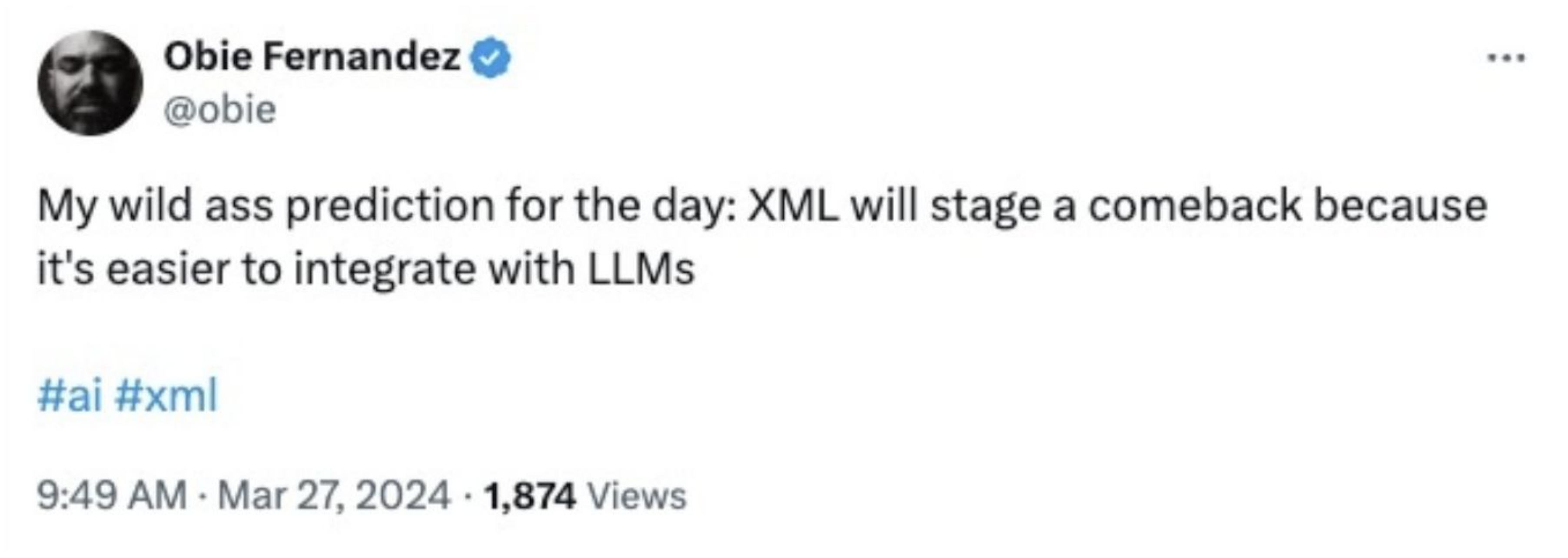
Structured IO Pattern

Structured Input
Helps the LLM better understand the task at hand and the specific data it needs to process

- Provide input data to the LLM in a structured standard format such as XML or JSON.
- Clearly delineate different parts of the prompt, such as instructions, examples, and data

Structured Output
Simplifies the parsing and integration of responses into your application's workflow.

- Use specific tags or delimiters to mark different parts of the output, such as entity names, values, or categories
- Enables easy extraction of relevant information from the response



Obie Fernandez
@obie

My wild ass prediction for the day: XML will stage a comeback because it's easier to integrate with LLMs

#ai #xml

9:49 AM · Mar 27, 2024 · **1,874** Views

Here's an example of how you can use XML tags to structure input when asking an LLM to extract attributes from a product description:

```
<description>
The SmartHome Mini is a compact smart home assistant available in black or
white for only $49.99. At just 5 inches wide, it lets you control lights,
thermostats, and other connected devices via voice or app—no matter where you
place it in your home. This affordable little hub brings convenient
hands-free control to your smart devices.
</description>

Extract <name>, <size>, <price>, and <color> from this product <description>.
```

By using XML tags to structure the input and output, the LLM implicitly understands that it should generate a response in XML:

```
<name>SmartHome Mini</name>
<size>5 inches wide</size>
<price>$49.99</price>
<color>black or white</color>
```

Ventriloquist Pattern

Allows you to guide the AI’s responses by preloading hardcoded user-assistant exchanges into the conversation transcript before starting any completions.

1. Plan the desired outcome for the AI’s responses
2. Create a set of hardcoded user-assistant exchanges that guide the AI towards the intended direction
3. Preload these exchanges into the conversation transcript before starting any completions
4. Initiate the chat completion process
5. Response continues where the hardcoded exchanges left off

```ruby
class AlternateKeywords
  include Raix::ChatCompletion

  PROMPT = <<~END
    Matching the original language of the question generate 3 alternate
    keywords that might produce better results. Reply with just the list,
    one per line.
  END

  def call(question)
    transcript << { system: "You are a powerful web search engine" }
    transcript << { user: question }
    transcript << { assistant: "Searching... no results found." }
    transcript << { user: PROMPT }
    chat_completion
  end

  def max_tokens
    30
  end

  def model
    [
      "databricks/dbrx-instruct:nitro",
      "cohere/command-r",
    ]
  end
end
```

```ruby
PROMPT = <<~END
  Matching the original language of the question generate 3 alternate
  keywords that might produce better results. Reply with just the list,
  one per line.
END

def call(question)
  transcript << { system: "You are a powerful web search engine" }
  transcript << { user: question }
  transcript << { assistant: "Searching... no results found." }
  transcript << { user: PROMPT }
  chat_completion
end
```

Discrete Components

Individually separate and distinct AI-based building blocks

Predicate Pattern

Pose a specific question to the AI model and expect a definitive yes or no answer.

1. Formulate a specific question that can be answered with a yes or no response.
2. Provide the AI model with the relevant context or information needed to answer the question.
3. Prompt the AI model with the question and the provided context, expecting a binary response (and optional rationale or explanation for the answer.)
4. Use the response to determine the appropriate course of action.

```
BEGIN TRANSCRIPT
%{text}
END TRANSCRIPT

The assistant is a personal AI that can help with a wide range of tasks
or simply converse with the user in a friendly manner like a companion.
The assistant has a long-term memory facility which it can invoke to
remember things that the user has told it in the past.

First analyze what the user is talking about, particularly the subjects
involved, which may include people, places, things, or concepts that are
personal to them and not in the public domain or your base knowlege.

Then answer the question:
Does the transcript contain enough context for the assistant to be able
to answer the user question without making assumptions about what they're
talking about?

Your response must begin with 'Yes, ' or 'No, ' followed by your
rationale. If you answer no, then the assistant will search its long-term
memory to gather more context, but that is an expensive operation and
should be avoided if possible.
```

API Facade Pattern

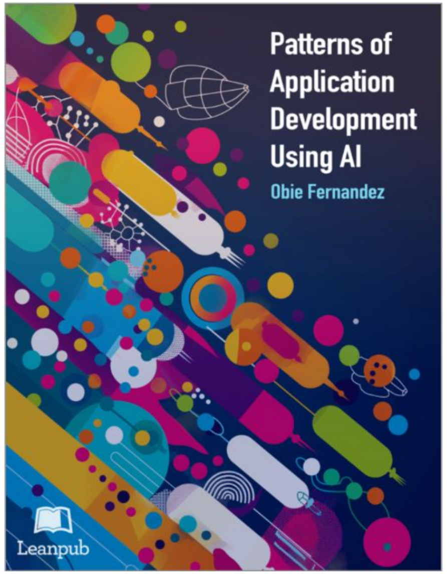
Sits alongside a more generalized AI assistant providing access to an API via a single plaintext request function.

1. Single request endpoint accepting plaintext or parameterized input.
2. Handles requests as *looped chat completion* with access to functions that map to needed API endpoints.
3. Returns a response that incorporates the results of the function calls.

```
SYSTEM_DIRECTIVE = <<~END
  The "user" conversing with you is another GPT helping its human user to
  manage their Gmail account. Because your replies will be processed by
  another AI, they do not need conversational commentary. Include full
  message data (id, recipients, subject line, full body) in responses about
  specific email messages.
END
```

Books Bundles Courses Tracks Search Leanpub

Patterns of Application Development Using AI



This book is 97% complete

LAST UPDATED ON 2024-06-29

Obie Fernandez

$39.00
MINIMUM PRICE

$69.00
SUGGESTED PRICE

YOU PAY
$69.00

AUTHOR EARNS
$55.20

YOU PAY IN US $
$69.00

EU customers: Price excludes VAT. VAT is added during checkout.

Add Ebook to Cart

Add to Wish List

Read Free Sample

Table Of Contents