Fine Tune your personal LLM assistant to Secure coding Workshop

Or Sahar & Yariv Tal, Secure From Scratch

2025

1 Information & Preparation

- Duration - 4 hours
- Attendees - 30 Max
- Categories - Machine Learning and Security;Web Application Security
- Attendees will actively engage by designing prompts for different coding scenarios, fine-tuning their own LLMs, and building custom assistants. Each participant will work hands-on to create, test, and iterate on their own models and use cases.
- What knowledge, skills, or experience should the students have prior to the workshop: Basic Python programming skills
- **Extremely Important:** You must set up OpenAI API Access in advance (more details in this below)
- Hardware Requirements:
  - Laptop with at least 8GB RAM (16GB recommended)
  - Stable internet connection
  - Admin rights to install packages/tools
- Software Requirements:
  - Python 3.8+
  - IDE (e.g., VS Code, PyCharm)
  - Git
  - OpenAI Python SDK (openai)

1.1 Steps to Set Up OpenAI API Access

1. Sign Up or Log In
   - Visit `https://openai.com/api` and click Start building.
   - Sign in or create an account. Verify your email and phone number when prompted.
2. Enable Billing
   - Go to the left-hand menu: Billing → Overview → Set up paid account.
   - Add a payment method and make an initial payment (minimum $5) to enable API usage.
3. (Optional) Set a Usage Cap
   - Navigate to Billing → Usage limits.
   - Define a soft and hard monthly limit to control expenses during the workshop.

4. Create an API Key

   - Go to the API Keys page and click Create new secret key.
   - Copy the key immediately; it will not be shown again.
   - Store it securely using an environment variable:

     ```
     export OPENAI_API_KEY="sk-..."
     ```

5. (Optional) Use Projects and Service Account Keys

   - Navigate to **Projects** → **Create**.
   - Generate service-account keys if you want isolated usage quotas for exercises.

6. Sanity Test

   ```
   curl https://api.openai.com/v1/models \
   -H "Authorization: Bearer $OPENAI_API_KEY"
   ```

   - A successful response should return a list of models.

1.2 Keep the treasure

- Never hardcode or share API keys. Use environment variables or secure storage.
- Enable multi-factor authentication (MFA) for the OpenAI account.
- Use separate keys per project and restrict endpoints where possible.
- Monitor **Billing** → **Usage** to track consumption during the workshop.

2 Abstract

In today's landscape, generative AI coding tools are powerful but often insecure, raising concerns for developers and organizations alike. This hands-on workshop will guide participants in building a secure coding assistant tailored to their specific security needs.

We'll begin by exploring the security limitations of current AI coding tools and discussing why fine-tuning is critical for secure development. Participants will then create and fine-tune their own LLM-based assistants using provided examples and their own use cases. By the end of the session, each attendee will have a functioning, security-focused AI coding assistant and a clear understanding of how to improve it further.

3 Outline

1. Introduction & Security Landscape (15 min)
    - Workshop goals and flow
    - Risks of AI-generated code
2. Prompt Engineering for Secure Code (30 min)
    - Principles of secure prompt design
    - Live prompt testing with OpenAI
3. Building a Basic Secure Assistant (30 min)
    - Using an API to generate and validate code
    - Adding simple checks or filters
4. Fine-Tuning Overview & Prep (20 min)
    - When to fine-tune
    - Dataset format (JSONL), small demo
5. Hands-On: Fine-Tuning (45 min)
    - Create a minimal dataset
    - Run a fine-tuning job
    - Quick test on the tuned model
6. (An) Interactive Assistant (60 min)
    - Build a CLI or lightweight interface in Python
7. Wrap-Up & Next Steps (20 min)
    - Summary
    - Best practices
    - Q&A

4 The Workshop

4.1 Workshop goals and flow

These are the goals of the workshop:

- Customize our llm for secure code generation
- Cover security use-cases that are more complicated then merely prompting for secure code
- Establish a methodology that will continute beyond the workshop for ongoing llm fine-tuning
- Create a community that will share prompts for secure code generation

4.2 The flow

After the introduction, we'll cover various security-related use cases: explain them, exploit them, learn how to write prompts that return secure versions, and verify those versions.

Next, we'll create an assistant containing all the prompt instructions we gathered.

Finally, we'll fine-tune our model to work without the assistant.

Each phase will include a demonstration followed by individual hands-on practice.

4.3 Risk of AI-generated code

Is there really a problem? Does Generative AI generate insecure code?

The article "Lost at C: A User Study on the Security Implications of LLM Code Assistants" [1] describes research conducted on the risks of using coding assistants. In the study, a group of computer science students was given a set of tasks. Some were allowed to use code assistants, while others were not.

The conclusion was that the use of GenAI did not increase the number of vulnerabilities in the code. However, developers tended to accept the assistant's suggestions blindly—meaning that if vulnerable code was suggested, they were likely to use it without making any corrections.

The experiment was conducted in the C programming language and focused on common vulnerabilities such as use-after-free, null pointer dereference, and out-of-bounds writes.

Another experiment [3] found that 16 out of 21 security-related prompts resulted in a vulnerable code snippet.

Yet another study [2] found that 40% of Copilot-generated programs contained dangerous vulnerabilities.

In conclusion, there **is** a problem: generative AI **does** generate insecure code.

4.4 Prompt Engeering For Secure coding

4.4.1 Secure Coding Priciples

PREVENT is an easy-to-remember acronym for secure coding prinicples:

Priority, **P**rivacy, **P**ermissions: Security is the first priority

Redundant defenses ensure multilayerd defense in depth

Easy to use securely. Easy to use safely. Always prefer safe by default to safe through action

Verify if the components you use and your implementation are safe and whether they follow current best practices

Events, **E**rrors and **E**xceptions - Log them, manage them

Neat code. Security through clarity. Neat code is not just easy to maintain, but also reduces undetected vulnerabilities.

Trust boundaries. Transferring data or control between entities with different trust levels requires pro-active action.

4.4.2 Prompting for Secure Coding

- Always ask for secure code — sometimes, simply adding the word "secure" can make a difference.
- If you are aware of specific risks (e.g., secure file upload), be as specific as possible in your prompt.
- After receiving the code, ask the model to identify any vulnerabilities in the generated output.
- Ask the model to fix the identified vulnerabilities.
- Ask the model to fix the vulnerabilities again — it often misses issues in the first pass.
- Compare the final version with the previous one; the model may fix one issue while unintentionally reintroducing or omitting others from earlier iterations.

4.4.3 Security related use cases / Best practice / Vulnerabilies

For the following demo we used gpt4.1, please be aware that the answer isn't always the same.

Secure File Upload

In file upload funcioanlity we have to validate the following:

- File size - to prevent denial of service
- File extension
- File Magic bytes (the fixed signature at the file's start) - reveal the real file format
- File path - we'd better not trust the user contribued path

Demo

1. A plain prompt for a code snippet:
   develop a file upload method in c#

   The snippet:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.IO;
using System.Threading.Tasks;

public class FileUploadController : Controller
{
  [HttpGet]
  public IActionResult Index()
  {
    return View();
  }

  [HttpPost]
  public async Task<IActionResult> UploadFile(IFormFile file)
  {
    if (file == null || file.Length == 0)
    return Content("No file selected");

    var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/uploads", file.FileName);

    // Ensure the directory exists
    Directory.CreateDirectory(Path.GetDirectoryName(path));

    using (var stream = new FileStream(path, FileMode.Create))
    {
      await file.CopyToAsync(stream);
    }

    return Content("File uploaded successfully!");
  }
}
```

2. Prompting for a secure snippet:
   develop a secure file upload method in c#

   The snippet:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.IO;
using System.Threading.Tasks;

[Route("api/[controller]")]
[ApiController]
public class FileUploadController : ControllerBase
{
  private readonly string[] permittedExtensions = { ".txt", ".jpg", ".png", ".pdf" };
  private const long maxFileSize = 10 * 1024 * 1024; // 10 MB

  [HttpPost("upload")]
  public async Task<IActionResult> UploadFile(IFormFile file)
  {
    if (file == null || file.Length == 0)
    return BadRequest("No file uploaded.");

    // Check file size
    if (file.Length > maxFileSize)
    return BadRequest("File size exceeds the limit.");

    // Calculate hash of the file or validate further if needed

    // Check file extension
    var ext = Path.GetExtension(file.FileName).ToLowerInvariant();
    if (string.IsNullOrEmpty(ext) || Array.IndexOf(permittedExtensions, ext) < 0)
    {
      return BadRequest("Invalid file extension.");
    }

    var filePath = Path.Combine("uploads", Guid.NewGuid().ToString() + ext);

    // Save the file
    using (var stream = System.IO.File.Create(filePath))
    {
      await file.CopyToAsync(stream);
```

```
      }

      return Ok(new { filePath });
    }
  }
```

This is almost what we need - but it doesn't include magic bytes testing.

An attacker can rename *evil.exe* to *cute.jpg*; the browser and server both see ".jpg", yet the content is still executable.

3. After we put some more thought and effort into the prompt:
   develop a secure file upload method in c#, make sure to check the magic bytes

   The snippet:

```
using Microsoft.AspNetCore.Http;
using System;
using System.IO;
using System.Threading.Tasks;
using System.Collections.Generic;

// Magic Bytes for common types
public static class MagicBytes
{
  public static readonly Dictionary<string, byte[]> FileSignatures = new()
  {
    // JPG
    { ".jpg", new byte[] { 0xFF, 0xD8, 0xFF } },
    // PNG
    { ".png", new byte[] { 0x89, 0x50, 0x4E, 0x47 } },
    // PDF
    { ".pdf", new byte[] { 0x25, 0x50, 0x44, 0x46 } },
    // ZIP
    { ".zip", new byte[] { 0x50, 0x4B, 0x03, 0x04 } }
  };

  public static bool HasValidSignature(string extension, Stream fileStream)
  {
    if (!FileSignatures.ContainsKey(extension.ToLower())) return false;
    var signature = FileSignatures[extension.ToLower()];
    var buffer = new byte[signature.Length];

    fileStream.Position = 0;
    fileStream.Read(buffer, 0, buffer.Length);

    fileStream.Position = 0;
    for (int i = 0; i < signature.Length; i++)
    {
      if (buffer[i] != signature[i]) return false;
    }
    return true;
  }
}

public class FileUploadHelper
{
  private static readonly string[] permittedExtensions = { ".jpg", ".png", ".pdf", ".zip" };
  private const long MAX_FILE_SIZE = 10 * 1024 * 1024; // 10 MB

  public static async Task<string> UploadAsync(IFormFile file, string uploadPath)
  {
    if (file == null || file.Length == 0)
    throw new ArgumentException("File is empty");
    if (file.Length > MAX_FILE_SIZE)
    throw new ArgumentException("File too large");

    var ext = Path.GetExtension(file.FileName).ToLowerInvariant();

    if (string.IsNullOrEmpty(ext) || Array.IndexOf(permittedExtensions, ext) < 0)
    throw new ArgumentException("Invalid file extension");

    // Check magic bytes
    using var stream = file.OpenReadStream();
    if (!MagicBytes.HasValidSignature(ext, stream))
    throw new ArgumentException("File signature does not match file extension.");

    // Generate a safe name
    var safeFileName = $"{Guid.NewGuid().ToString()}{ext}"; //$ is string interpolation
    var savePath = Path.Combine(uploadPath, safeFileName);

    Directory.CreateDirectory(uploadPath);
    using var output = File.Create(savePath);
```

```
    await stream.CopyToAsync(output);

    return safeFileName; // Return the stored file name.
  }
}
```

Ohhh no! The function is still not secure!

- The helper only verifies the outer file's magic bytes PK\x03\x04, confirming it is *a valid ZIP*, but never inspects the archive's contents.
- An attacker can therefore upload a ZIP containing evil.exe, macro-laden documents, gadget payloads, or a zip-bomb without detection.

Why this matters

- Type-filter bypass: malicious code slips through a filter meant to permit only benign formats.
- Accidental execution: any later extraction (by backend logic or an end-user) may run the payload.
- Scanning gaps: some AV/WAF setups skip or inadequately scan embedded files.
- Resource exhaustion: zip-bombs can consume CPU, RAM, or disk on decompression.

Hardening recommendations

(a) If ZIP uploads are not strictly required, remove .zip from permittedExtensions.

(b) If ZIPs are required:

i. Open the archive with System.IO.Compression.ZipArchive and iterate over entries.

ii. Reject any entry whose name ends with forbidden extensions (`.exe`, .dll, .js, ...) or uses double-extension tricks (.jpg.exe).

iii. Block path traversal (`../`) and symlink entries.

iv. Enforce limits on total uncompressed size, entry count, and nesting depth to stop zip-bombs.

v. Store the file without auto-extracting; route it through AV or content-disarm mechanisms before distribution.

SSRF (Server-Side Request Forgery)

The application fetches remote resources using user-supplied URLs without proper validation, allowing attackers to trick the server into sending requests to internal services or sensitive endpoints.

To mitigate this vulnerability, implement strict URL validation:

- Enforce https:// scheme only.
- Resolve hostnames to IP addresses and validate them before making the request.

- Maintain a welcome-list of trusted domains and block all others.
- Maintain a welcome-list of IPs that don't include private, loopback, and link-local IP ranges.

Demo

1. A plain prompt for a code snippet:
   Write a C# API method that displays a website preview for a user-supplied URL

   The snippet:

```
using Microsoft.AspNetCore.Mvc;
using System.Net.Http;
using System.Threading.Tasks;
using System.Text.RegularExpressions;
using System.Web;

[ApiController]
[Route("api/[controller]")]
public class WebsitePreviewController : ControllerBase
{
  [HttpGet]
  public async Task<IActionResult> GetPreview([FromQuery] string url)
  {
    if (string.IsNullOrWhiteSpace(url))
    return BadRequest("URL required.");

    // Basic URL validation; you might want to use a more robust approach
    if (!Uri.TryCreate(url, UriKind.Absolute, out var uriResult))
    return BadRequest("Invalid URL.");

    try
    {
      using var client = new HttpClient();
      client.DefaultRequestHeaders.UserAgent.ParseAdd("Mozilla/5.0 (compatible)");
      var html = await client.GetStringAsync(url);

      var title = Regex.Match(html, @"<title>\s*(.+?)\s*</title>", RegexOptions.IgnoreCase).Groups[1]?.Value ?? "";
      var descMatch = Regex.Match(html, @"<meta\s+name=[""']description[""']\s+content=[""'](.+?)[""']",
     ↪ RegexOptions.IgnoreCase);
      var ogTitleMatch = Regex.Match(html, @"<meta\s+property=[""']og:title[""']\s+content=[""'](.+?)[""']",
     ↪ RegexOptions.IgnoreCase);
      var ogDescMatch = Regex.Match(html, @"<meta\s+property=[""']og:description[""']\s+content=[""'](.+?)[""']",
     ↪ RegexOptions.IgnoreCase);
      var ogImgMatch = Regex.Match(html, @"<meta\s+property=[""']og:image[""']\s+content=[""'](.+?)[""']",
     ↪ RegexOptions.IgnoreCase);

      var preview = new
      {
        url,
        title = HttpUtility.HtmlDecode(ogTitleMatch.Groups[1].Value) ?? HttpUtility.HtmlDecode(title),
        description = HttpUtility.HtmlDecode(ogDescMatch.Groups[1].Value) ??
        HttpUtility.HtmlDecode(descMatch.Groups[1].Value),
        image = ogImgMatch.Groups[1].Value
      };

      return Ok(preview);
    }
    catch (HttpRequestException)
    {
      return NotFound("Could not fetch the URL.");
    }
    catch
    {
      return StatusCode(500, "An error occurred while processing the URL.");
    }
  }
}
```

   This code is vulnerable to SSRF (Server-Side Request Forgery) because it fetches the content of a user-supplied URL without enforcing any restrictions or validation on the destination. An attacker can exploit this by providing internal or sensitive URLs (e.g., 'http://localhost:5000/admin', 'http://169.254.169.254') to trick the server into making requests on their behalf. Since the server performs the request, it may reach internal services not exposed to the public, potentially leaking sensitive data or enabling further attacks such as internal port scanning or access to metadata services.

2. Prompting for a secure snippet:
   Write a secure C# API method that displays a website preview for a user-supplied URL

   The snippet:

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Sockets;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using HtmlAgilityPack;
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("api/[controller]")]
public class PreviewController : ControllerBase
{
  private static readonly HttpClient httpClient = new HttpClient
  {
    Timeout = TimeSpan.FromSeconds(5) // Short timeout
  };

  [HttpGet("website-preview")]
  public async Task<IActionResult> GetWebsitePreview([FromQuery] string url)
  {
    try
    {
      // 1. Validate URL format and scheme
      if (!Uri.TryCreate(url, UriKind.Absolute, out var uri) ||
      (uri.Scheme != Uri.UriSchemeHttp && uri.Scheme != Uri.UriSchemeHttps))
      {
        return BadRequest("Invalid URL.");
      }

      // 2. Prevent SSRF: Block localhost, private IPs, etc.
      if (IsLocalOrPrivateAddress(uri))
      {
        return BadRequest("Forbidden URL.");
      }

      // 3. Fetch the page content securely
      var response = await httpClient.GetAsync(uri);
      if (!response.IsSuccessStatusCode)
      {
        return BadRequest("Failed to retrieve the website.");
      }

      if (response.Content.Headers.ContentType?.MediaType != "text/html")
      {
        return BadRequest("URL is not a HTML page.");
      }

      string html = await response.Content.ReadAsStringAsync();

      // 4. Parse and extract preview (title, description)
      var doc = new HtmlDocument();
      doc.LoadHtml(html);

      string title = doc.DocumentNode.SelectSingleNode("//title")?.InnerText?.Trim() ?? "";
      string description = doc.DocumentNode
      .SelectSingleNode("//meta[@name='description']")?
      .GetAttributeValue("content", "") ?? "";

      return Ok(new
      {
        Title = title,
        Description = description
      });
    }
    catch (Exception ex)
    {
      // Log ex (not shown here)
      return BadRequest("Error processing the URL.");
    }
  }

  // Prevent SSRF: block localhost, internal IPs, etc.
  private bool IsLocalOrPrivateAddress(Uri uri)
  {
    try
    {
      var host = uri.Host;

      // Block 'localhost' and common local hostnames
      if (host.Equals("localhost", StringComparison.OrdinalIgnoreCase)
      || host.Equals("127.0.0.1")
      || host.StartsWith("::1")
      || host.EndsWith(".local")
```

```
            )
            {
              return true;
            }

            // Attempt to resolve DNS
            var addresses = Dns.GetHostAddresses(host);
            foreach (var ip in addresses)
            {
              if (IPAddress.IsLoopback(ip)) return true;

              // Block private address ranges: 10/8, 172.16/12, 192.168/16, IPv6 unique local etc.
              if (ip.AddressFamily == AddressFamily.InterNetwork)
              {
                byte[] bytes = ip.GetAddressBytes();
                if (bytes[0] == 10
                || (bytes[0] == 172 && bytes[1] >= 16 && bytes[1] <= 31)
                || (bytes[0] == 192 && bytes[1] == 168))
                return true;
              }
              else if (ip.AddressFamily == AddressFamily.InterNetworkV6)
              {
                if (ip.IsIPv6LinkLocal || ip.IsIPv6SiteLocal)
                return true;
              }
            }
          }
          catch
          {
            return true; // On DNS failure, err on the safe side
          }
          return false;
        }
      }
```

The code is still vulnerable to SSRF

- DNS-rebind (TOCTOU, Time to check time to use) – the host is resolved during validation, but HttpClient resolves it again when connecting; an attacker can return a public IP first and an internal IP seconds later, diverting the real request.
- Automatic redirects – HttpClient follows 3xx responses by default, so a harmless URL can redirect the server to http://169.254.169.254/ or another internal endpoint after the initial check.
- Non-canonical IP forms – dotless integers, octal/hex, mixed IPv6 representations, or broadcast addresses are not recognised by the filter yet resolve to loopback-/private space.
- Port abuse – even when the IP is external, the code allows requests to sensitive ports; there is no allow/deny list for outbound ports.

Full mitigation requires disabling auto-redirects, validating the final resolved IP after each hop, using the same resolved address for the actual socket connection, and enforcing strict host and port allow-lists.

Race Condtion

According to cwe.org [4]:

"The product checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the product to perform invalid actions when the resource is in an unexpected state."

"This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs."

Demo

1. A plain prompt for a code snippet:
   Write C# code that exposes an API endpoint to apply a coupon to an order

   - *Accept orderId and couponCode as input.*
   - *Look up the coupon and verify it has not been used.*
   - *If unused, reduce the order total by 50%.*
   - *Mark the coupon as used so it cannot be applied again.*

   The snippet:

```
public class Order
{
  public int Id { get; set; }
  public decimal Total { get; set; }
}

public class Coupon
{
  public string Code { get; set; }
  public bool IsUsed { get; set; }
}
```

```
public static class DataStore
{
  public static List<Order> Orders = new List<Order>
  {
    new Order { Id = 1, Total = 100 },
    new Order { Id = 2, Total = 50 }
  };

  public static List<Coupon> Coupons = new List<Coupon>
  {
    new Coupon { Code = "HALFOFF", IsUsed = false },
    new Coupon { Code = "USEDONE", IsUsed = true }
  };
}
```

```
public class ApplyCouponRequest
{
  public int OrderId { get; set; }
  public string CouponCode { get; set; }
}
```

```
using Microsoft.AspNetCore.Mvc;

[ApiController]
[Route("api/[controller]")]
public class OrdersController : ControllerBase
{
  [HttpPost("apply-coupon")]
  public IActionResult ApplyCoupon([FromBody] ApplyCouponRequest request)
  {
    // 1. Lookup the order
    var order = DataStore.Orders.FirstOrDefault(o => o.Id == request.OrderId);
    if (order == null)
    return NotFound(new { Message = "Order not found." });

    // 2. Lookup the coupon
    var coupon = DataStore.Coupons.FirstOrDefault(c => c.Code == request.CouponCode);
    if (coupon == null)
    return NotFound(new { Message = "Coupon not found." });

    // 3. Check if coupon has been used
    if (coupon.IsUsed)
    return BadRequest(new { Message = "Coupon has already been used." });
```

```
        // 4. Apply coupon: 50% off
        order.Total *= 0.5m;

        // 5. Mark coupon as used
        coupon.IsUsed = true;

        return Ok(new
        {
          Message = "Coupon applied successfully.",
          NewTotal = order.Total
        });
      }
    }
```

The coupon's "check" ('if (coupon.IsUsed)') and "update" ('coupon.IsUsed = true' + discounted total) happen in separate, non-atomic operations with no lock or transaction; if two requests hit the endpoint almost simultaneously, both can see 'IsUsed == false', each apply the 50 % discount, and only then set the flag—allowing one coupon to be redeemed multiple times and corrupting the order totals.

2. Prompting for a secure snippet:
   Write secure C# code that exposes an API endpoint to apply a coupon to an order.

   - *Accept orderId and couponCode as input.*
   - *Look up the coupon and verify it has not been used.*
   - *If unused, reduce the order total by 50%.*
   - *Mark the coupon as used so it cannot be applied again.*

   The snippet:

```
    public class Order
    {
      public int Id { get; set; }
      public decimal Total { get; set; }
    }

    public class Coupon
    {
      public string Code { get; set; }
      public bool IsUsed { get; set; }
    }
```

```
  public static class DataStore
  {
    public static List<Order> Orders = new List<Order>
    {
      new Order { Id = 1, Total = 100 },
      new Order { Id = 2, Total = 50 }
    };

    public static List<Coupon> Coupons = new List<Coupon>
    {
      new Coupon { Code = "HALFOFF", IsUsed = false },
      new Coupon { Code = "USEDONE", IsUsed = true }
    };
  }
```

```
public class ApplyCouponRequest
{
  public int OrderId { get; set; }
  public string CouponCode { get; set; }
}
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

[ApiController]
[Route("api/[controller]")]
public class OrdersController : ControllerBase
{
  private readonly AppDbContext _db;

  public OrdersController(AppDbContext db)
  {
    _db = db;
  }

  [HttpPost("apply-coupon")]
  public async Task<IActionResult> ApplyCoupon([FromBody] ApplyCouponRequest request)
  {
    if (string.IsNullOrWhiteSpace(request.CouponCode))
    return BadRequest("Coupon code required.");

    // Validate order
    var order = await _db.Orders.FindAsync(request.OrderId);
    if (order == null)
    return NotFound("Order not found.");

    // Look up and validate coupon (use tracking for concurrency)
    var coupon = await _db.Coupons
    .Where(c => c.Code == request.CouponCode)
    .FirstOrDefaultAsync();

    if (coupon == null)
    return BadRequest("Invalid coupon code.");

    if (coupon.IsUsed)
    return BadRequest("Coupon already used.");

    // Apply 50% discount
    order.Total *= 0.5m;
    coupon.IsUsed = true;

    try
    {
      await _db.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
      return Conflict("Coupon application conflict. Try again.");
    }

    // Only return result, do not reveal coupon details.
    return Ok(new { orderId = order.Id, newTotal = order.Total });
  }
}
```

‘coupon.IsUsed‘ is still checked and updated in two separate steps, and nothing enforces atomicity: the ‘Coupons‘ table lacks a concurrency token or conditional update, so two parallel transactions can both read ‘IsUsed = 0‘, each apply the 50 % discount, set the flag, and commit successfully—no ‘DbUpdateConcurrencyException‘ is raised because EF only detects conflicts on properties marked with ‘[ConcurrencyCheck]‘ or a ‘rowversion‘ column. Database-level protection (e.g. a single ‘UPDATE ... SET IsUsed = 1 WHERE Code = @code AND IsUsed = 0‘) or a row-version column is required to truly close the race window.

4.4.4 Lab Time - 30 min

Now, let’s put our knowledge into practice

1. Go to OpenAI’s website: openai.com
2. Log in!
3. Go to the playground: platform.openai.com/playground
4. Try the file upload prompts

- Use a naive prompt ("Develop a file upload method in language_of _your choice)
- Bonus - Integrate it into a project & Exploit it!
- Add the word "secure"
- Add a full security-aware file upload prompt

5. Repeat what you did in the previous step for SSRF, Race Condition and / or a scenario of your own choosing

4.5 Building a secure assistant

One approach to improving developer security is configuring an AI assistant with security-related instructions embedded in its system prompt. When done correctly, this ensures that developers receive only secure code suggestions.

4.5.1 Demo

The Demo was conducted in Open AI platform: `https://platform.openai.com/assistants`

Secure File Upload

System prompt:

You are a helpful assistant that writes secure csharp code.
Please respond with a secure csharp code snippet whenever prompted. Please provide only code snippet, no explanations.
in file upload please make sure the extension and the mime type of the file is allowed. make sure the magic bytes are allowed. make sure the path is within allowed folder, the file is not too heavy. make sure that incoming files don't affect or delete any existing files on the server. give the file a new unique name using guid.

User prompt:

Develop a file upload method in c#

The result:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;

public class SecureFileUploader
{
  private readonly string[] allowedExtensions = { ".jpg", ".jpeg", ".png", ".pdf" };
  private readonly (byte[] header, string extension)[] magicBytes =
  {
    (new byte[] { 0xFF, 0xD8, 0xFF }, ".jpg"),
    (new byte[] { 0x89, 0x50, 0x4E, 0x47 }, ".png"),
    (new byte[] { 0x25, 0x50, 0x44, 0x46 }, ".pdf")
  };
  private const long maxFileSize = 5 * 1024 * 1024; // 5 MB
  private readonly string allowedDirectory = Path.GetFullPath("Uploads");

  public async Task<string> UploadAsync(IFormFile file)
  {
    if (file == null || file.Length == 0 || file.Length > maxFileSize)
    throw new InvalidOperationException("File is missing or too large.");

    var extension = Path.GetExtension(file.FileName).ToLowerInvariant();

    if (Array.IndexOf(allowedExtensions, extension) < 0)
    throw new InvalidOperationException("File extension not allowed.");

    using (var fileStream = file.OpenReadStream())
    {
      var header = new byte[8];
      int readBytes = await fileStream.ReadAsync(header, 0, header.Length);
      bool validMagic = false;
```

```
        foreach (var (magic, ext) in magicBytes)
        {
          if (readBytes >= magic.Length && extension == ext)
          {
            bool match = true;
            for (int i = 0; i < magic.Length; i++)
            {
              if (header[i] != magic[i])
              {
                match = false;
                break;
              }
            }
            if (match)
            {
              validMagic = true;
              break;
            }
          }
        }

        if (!validMagic)
        throw new InvalidOperationException("Magic bytes do not match extension.");

        fileStream.Position = 0;

        string uniqueName = Guid.NewGuid().ToString("N") + extension;
        string uploadPath = Path.Combine(allowedDirectory, uniqueName);
        string fullPath = Path.GetFullPath(uploadPath);

        if (!fullPath.StartsWith(allowedDirectory))
        throw new UnauthorizedAccessException("Path traversal detected.");

        if (!Directory.Exists(allowedDirectory))
        Directory.CreateDirectory(allowedDirectory);

        using (var output = new FileStream(fullPath, FileMode.CreateNew, FileAccess.Write))
        {
          await fileStream.CopyToAsync(output);
        }

        return uniqueName; // or return fullPath if needed
      }
    }
  }
```

SSRF

System prompt:

1. URL Format & Scheme
   - Accept only well-formed, *absolute* URLs.
   - Permit https scheme exclusively; reject every other scheme.
2. Trusted Domains & Ports
   - Hostname must match (case-insensitive) a predefined trusted-domain allow-list.
   - Permit only explicitly welcomelisted ports (e.g. 443); reject all others.
3. Direct IP Access Restrictions
   - If the host is an IP address, reject it when it is:
     - Loopback
     - Private
     - Link-local
     - IPv4-mapped IPv6
4. Safe DNS Resolution

- Resolve the hostname *immediately* before each request (DNS timeout ≤ 3 s).
- Reject the request if *any* resolved IP fails the IP-safety rules.
- Prevent TOCTOU:
  - Pin the selected IP via a custom ConnectCallback.
  - Disable re-resolution by the HTTP client.
- Multi-IP fallback: if the first safe IP fails, retry sequentially with the next safe IP, each with its own pinned handler.

5. Redirect Handling
   - Follow redirects manually (maximum 5).
   - Re-validate scheme, hostname, port, and IP before each redirect hop.
6. Final IP Revalidation
   - Re-check the resolved IP immediately before streaming the response body.
7. TLS & Host Integrity
   - Enforce full TLS certificate validation; never disable or bypass it.
   - Preserve the original hostname in the Host header.
   - Ensure the SNI field matches the original hostname when IP-pinning.
8. Response Size & Parsing Limits
   - Reject responses larger than 1 MB.
   - Parse only the first 4 KB of HTML (or N bytes as configured) to mitigate ReDoS.
9. HTTP Client Configuration
   - Always set a static User-Agent header (e.g. PreviewBot/1.0).
   - Apply a strict overall request timeout (e.g. ≤ 10 s).
   - Do *not* enable automatic redirect following.

User prompt:

Write a C# API method that displays a website preview for a user- supplied URL.

The result:

```
using System;
using System.Buffers;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Net.Sockets;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
```

```
[ApiController]
[Route("api/[controller]")]
public class PreviewController : ControllerBase
{
  private static readonly HashSet<string> AllowedHosts = new(StringComparer.OrdinalIgnoreCase)
  {
    "example.com",
    "www.example.com",
    // Add more trusted domains here
  };

  private static readonly HashSet<int> AllowedPorts = new() { 443 };
  private const int MaxRedirects = 5;
  private const int MaxResponseBytes = 1024 * 1024; // 1MB
  private const int MaxParseBytes = 4 * 1024; // 4KB
  private static readonly TimeSpan DnsTimeout = TimeSpan.FromSeconds(3);
  private static readonly TimeSpan RequestTimeout = TimeSpan.FromSeconds(10);

  [HttpGet]
  public async Task<IActionResult> GetPreview([FromQuery] string url)
  {
    try
    {
      if (!TryValidateUrl(url, out Uri validatedUri, out string error))
      return BadRequest(new { error });

      int redirectCount = 0;
      Uri currentUri = validatedUri;

      HashSet<string> visitedUrls = new(StringComparer.OrdinalIgnoreCase);

      while (redirectCount <= MaxRedirects)
      {
        List<IPAddress> resolvedIps = await ResolveSafeIpsAsync(currentUri.Host);

        if (resolvedIps.Count == 0)
        return BadRequest(new { error = "No safe IPs found for hostname." });

        Exception lastException = null;

        foreach (IPAddress ip in resolvedIps)
        {
          using var cts = new CancellationTokenSource(RequestTimeout);
          try
          {
            // Custom SocketsHttpHandler per IP with IP pinning
            using var handler = CreatePinnedHandler(ip, currentUri.Host);

            using var client = new HttpClient(handler)
            {
              Timeout = Timeout.InfiniteTimeSpan // already have external timeout
            };

            client.DefaultRequestHeaders.UserAgent.ParseAdd("PreviewBot/1.0");
            client.DefaultRequestHeaders.Host = currentUri.Host;

            var request = new HttpRequestMessage(HttpMethod.Get, currentUri);
            request.Headers.Host = currentUri.Host;

            using var response = await client.SendAsync(request,
            HttpCompletionOption.ResponseHeadersRead, cts.Token);

            if (response.StatusCode == HttpStatusCode.MovedPermanently ||
            response.StatusCode == HttpStatusCode.Found ||
            response.StatusCode == HttpStatusCode.Redirect ||
            response.StatusCode == HttpStatusCode.TemporaryRedirect ||
            (int)response.StatusCode == 308)
            {
              if (++redirectCount > MaxRedirects)
              return BadRequest(new { error = "Too many redirects." });

              if (!response.Headers.Location.IsAbsoluteUri)
              currentUri = new Uri(new Uri(currentUri.GetLeftPart(UriPartial.Authority)), response.Headers.Location);
              else
              currentUri = response.Headers.Location;

              if (!TryValidateUrl(currentUri.ToString(), out validatedUri, out error))
              return BadRequest(new { error = "Unsafe redirect: " + error });

              if (!visitedUrls.Add(currentUri.AbsoluteUri))
              return BadRequest(new { error = "Redirect loop detected." });

              // Redo DNS/IP validation on each hop
              break; // leave foreach, process next hop
            }

            // Validate the remote endpoint's IP
            if (handler.ConnectedRemoteIp is not null && !IsSafeIp(handler.ConnectedRemoteIp))
            return BadRequest(new { error = "Unsafe endpoint after connect." });

            // Limit expected response size to 1 MB
            if (response.Content.Headers.ContentLength > MaxResponseBytes)
```

```
            return BadRequest(new { error = "Response too large." });

            await using var stream = await response.Content.ReadAsStreamAsync(cts.Token);

            // Stream only up to 1 MB
            byte[] buffer = ArrayPool<byte>.Shared.Rent(MaxParseBytes);
            int readLen = 0, totalRead = 0;
            using var ms = new MemoryStream();
            do
            {
              int toRead = Math.Min(MaxParseBytes - totalRead, buffer.Length);
              readLen = await stream.ReadAsync(buffer, 0, toRead, cts.Token);
              if (readLen > 0)
              {
                ms.Write(buffer, 0, readLen);
                totalRead += readLen;

                if (ms.Length > MaxResponseBytes)
                {
                  ArrayPool<byte>.Shared.Return(buffer);
                  return BadRequest(new { error = "Response exceeds 1 MB limit." });
                }
                if (totalRead >= MaxParseBytes) break;
              }
            }
            while (readLen > 0);

            ArrayPool<byte>.Shared.Return(buffer);

            // Minimal HTML preview parsing (only 4KB)
            ms.Position = 0;
            using var reader = new StreamReader(ms, leaveOpen: true);
            string htmlSnip = await reader.ReadToEndAsync();
            var title = ExtractHtmlTagContent(htmlSnip, "title");
            var metaDesc = ExtractMetaDescription(htmlSnip);

            return Ok(new
            {
              url = currentUri.ToString(),
              title,
              description = metaDesc,
            });
          }
          catch (Exception ex) when (!(ex is OperationCanceledException))
          {
            lastException = ex;
            continue; // try next IP
          }
        }

        if (++redirectCount > MaxRedirects)
        return BadRequest(new { error = "Too many redirects." });

        if (lastException is not null)
        return StatusCode(502, new { error = "Failed to fetch preview.", details = lastException.Message });

        break; // Success, don't loop redirects.
      }

      return BadRequest(new { error = "Invalid redirect logic." });
    }
    catch (OperationCanceledException)
    {
      return StatusCode(504, new { error = "Request timed out." });
    }
    catch (Exception ex)
    {
      return StatusCode(502, new { error = "Failed to fetch preview.", details = ex.Message });
    }
  }

  private static bool TryValidateUrl(string input, out Uri uri, out string error)
  {
    error = null;
    uri = null;

    if (string.IsNullOrWhiteSpace(input))
    {
      error = "URL cannot be empty.";
      return false;
    }

    if (!Uri.TryCreate(input, UriKind.Absolute, out uri))
    {
      error = "Malformed URL.";
      return false;
    }

    if (!string.Equals(uri.Scheme, "https", StringComparison.OrdinalIgnoreCase))
    {
      error = "Only https URLs are allowed.";
      return false;
```

```
        }

        if (!AllowedHosts.Contains(uri.Host))
        {
          error = "Domain is not allowed.";
          return false;
        }

        int port = uri.IsDefaultPort ? 443 : uri.Port;
        if (!AllowedPorts.Contains(port))
        {
          error = "Port is not allowed.";
          return false;
        }

        return true;
      }

      private static async Task<List<IPAddress>> ResolveSafeIpsAsync(string host)
      {
        using var cts = new CancellationTokenSource(DnsTimeout);
        var ips = new List<IPAddress>();
        try
        {
          var addresses = await Dns.GetHostAddressesAsync(host).WaitAsync(DnsTimeout, cts.Token);
          foreach (var ip in addresses)
          if (IsSafeIp(ip)) ips.Add(ip);
        }
        catch { }
        return ips;
      }

      private static bool IsSafeIp(IPAddress ip)
      {
        if (ip.AddressFamily != AddressFamily.InterNetwork && ip.AddressFamily != AddressFamily.InterNetworkV6)
        return false;

        if (IPAddress.IsLoopback(ip)) return false;

        // Match private (RFC1918, RFC4193), link-local
        if (ip.AddressFamily == AddressFamily.InterNetwork)
        {
          byte[] bytes = ip.GetAddressBytes();
          // 10.0.0.0/8
          if (bytes[0] == 10) return false;
          // 172.16.0.0/12
          if (bytes[0] == 172 && bytes[1] >= 16 && bytes[1] <= 31) return false;
          // 192.168.0.0/16
          if (bytes[0] == 192 && bytes[1] == 168) return false;
          // Link-local 169.254.0.0/16
          if (bytes[0] == 169 && bytes[1] == 254) return false;
        }
        else if (ip.AddressFamily == AddressFamily.InterNetworkV6)
        {
          if (ip.IsIPv4MappedToIPv6) return false;
          if (ip.IsIPv6LinkLocal || ip.IsIPv6SiteLocal || ip.IsIPv6Multicast) return false;
          if (ip.Equals(IPAddress.IPv6Loopback)) return false;
          // Unique local address fc00::/7
          byte[] bytes = ip.GetAddressBytes();
          if ((bytes[0] & 0xFE) == 0xFC) return false;
        }
        return true;
      }

      private static SocketsHttpHandlerWithIpPinning CreatePinnedHandler(IPAddress ip, string sniHost)
      {
        var handler = new SocketsHttpHandlerWithIpPinning(ip, sniHost)
        {
          AllowAutoRedirect = false,
          AutomaticDecompression = DecompressionMethods.None,
          UseCookies = false,
          ConnectTimeout = RequestTimeout
        };
        handler.SslOptions.RemoteCertificateValidationCallback = (sender, cert, chain, errors) =>
        errors == SslPolicyErrors.None;
        return handler;
      }

      private static string ExtractHtmlTagContent(string html, string tag)
      {
        var rx = new Regex($@"<\s*{tag}\s*[^>]*>(.*?)<\s*/\s*{tag}\s*>", RegexOptions.IgnoreCase | RegexOptions.Singleline
    ↪ );
        var match = rx.Match(html);
        return match.Success ? WebUtility.HtmlDecode(match.Groups[1].Value.Trim()) : null;
      }

      private static string ExtractMetaDescription(string html)
      {
        var rx = new Regex(@"<meta\s+[^>]*name=['""]?description['""]?[^>]*content=['""]?(.*?)['""]?[^>]*>", RegexOptions.
    ↪ IgnoreCase | RegexOptions.Singleline);
        var match = rx.Match(html);
        return match.Success ? WebUtility.HtmlDecode(match.Groups[1].Value.Trim()) : null;
```

```
    }

    // Custom handler to enforce connect to given IP, preserve Host/SNI, and expose remote connected IP
    private sealed class SocketsHttpHandlerWithIpPinning : SocketsHttpHandler
    {
      private readonly IPAddress _ip;
      private readonly string _sniHost;
      public IPAddress ConnectedRemoteIp { get; private set; }

      public SocketsHttpHandlerWithIpPinning(IPAddress ip, string sniHost)
      {
        _ip = ip;
        _sniHost = sniHost;

        ConnectCallback = async (ctx, ct) =>
        {
          var socket = new Socket(_ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
          try
          {
            using var cts = CancellationTokenSource.CreateLinkedTokenSource(ct, new CancellationTokenSource(
↪ RequestTimeout).Token);
            await socket.ConnectAsync(new IPEndPoint(_ip, ctx.DnsEndPoint.Port), cts.Token);
            if (!socket.Connected)
            throw new IOException("Failed to connect IP pinned socket.");
            ConnectedRemoteIp = ((IPEndPoint)socket.RemoteEndPoint).Address;
            var ns = new NetworkStream(socket, ownsSocket: true);

            var ssl = new SslStream(ns, leaveInnerStreamOpen: false, null, null, EncryptionPolicy.RequireEncryption);
            var sslOpts = new SslClientAuthenticationOptions
            {
              TargetHost = _sniHost, // SNI
              CertificateRevocationCheckMode = X509RevocationMode.Online,
              ApplicationProtocols = new List<SslApplicationProtocol> { SslApplicationProtocol.Http11,
↪ SslApplicationProtocol.Http2 }
            };
            await ssl.AuthenticateAsClientAsync(sslOpts, cts.Token);
            return ssl;
          }
          catch
          {
            socket.Dispose();
            throw;
          }
        };
      }
    }
  }
```

4.5.2 Lab Time - 30 min

Now, let's put our knowledge into practice

1. Go to OpenAI website: openai.com
2. Log in!
3. Go to assistants: platform.openai.com/assistants
4. Create a new assistant that is safe from unrestricted file uploads and SSRF.
5. Use the provided simple prompts to ensure the assistant does not generate insecure code snippets for developers who are not security-aware.
6. Repeat what you did in the previous step for Race Condition and / or a scenario of your own choosing

4.5.3 Limitations of Using Assitants for Secure Coding

Embedding all organization-specific security guidelines directly into the assistant's system prompt can contribute to secure code generation for developers. However, this approach has clear drawbacks: longer prompts increase both latency and API cost, and they reduce the available output space, risking truncation of the assistant's response.

4.6 Fine-Tuning

An effective approach for building secure coding assistants at scale is fine-tuning. Instead of relying solely on long system prompts, organizations can train a custom model using their own dataset of prompts and secure code completions. This method enables the assistant to learn how to consistently respond in a secure manner to common development queries.

Fine-tuning is especially useful in organizational settings, where application security teams and developers can collaborate to build a shared dataset tailored to their specific coding standards and threat model. Once the dataset is ready, a fine-tuned assistant can be created and made available across the organization to provide fast, consistent, and secure code suggestions.

4.6.1 Steps for Conducting Fine-Tuning with OpenAI

1. Identify one or more security-focused use cases relevant to your organization.
2. Write multiple prompt variations that reflect how developers in your organization naturally phrase requests for these scenarios.
3. Create secure code snippets that address each use case.
4. Compile all prompt–snippet pairs into a properly formatted .jsonl file. Ensure proper escaping: use `\n` for newlines, escape double quotes with a backslash (`\"`), and escape backslashes themselves as `\\`.
5. Initiate a fine-tuning job on the OpenAI platform and provide the prepared .jsonl dataset.
6. Test your fine-tuned assistant using the OpenAI playground or API with representative prompts.
7. Iterate: refine prompts and snippets based on assistant output quality, then re-tune if needed.

4.6.2 Demo

The structure of the json file:

```
{
  "messages": [
    {
      "role": "user",
      "content": "Can you write a file upload API method in .NET Core"
    },
    {
      "role": "assistant",
      "content": "using System.IO;\nusing System.Threading.Tasks;....."
    }
  ]
}
```

The results of using the fine tuned model with file upload use case:

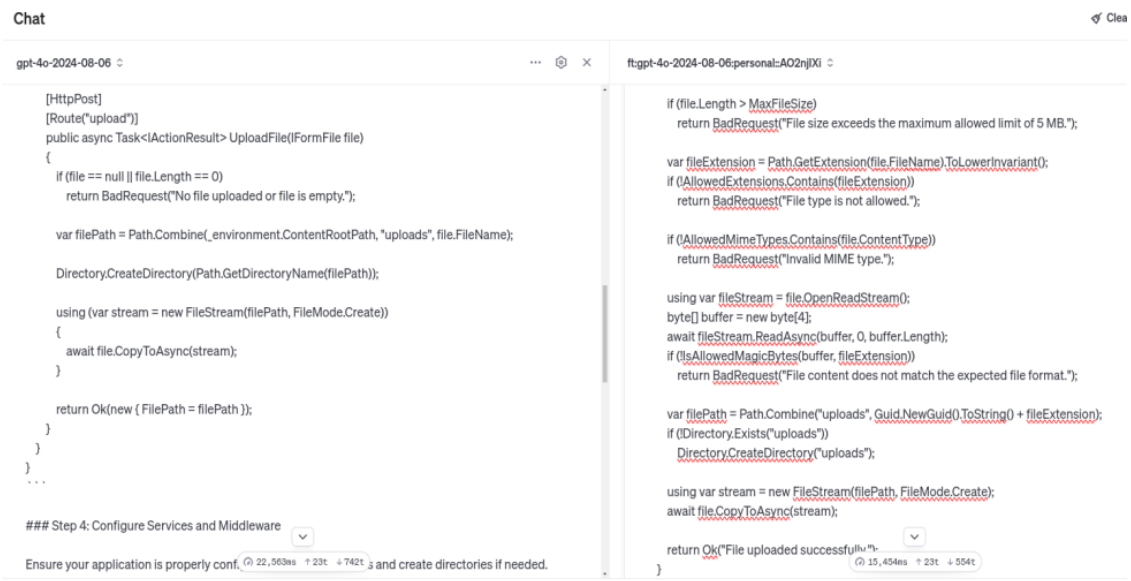

Figure 1: Secure file upload snippet

4.6.3 Lab Time - 40 min

Now, let's put our knowledge into practice

1. Go to OpenAI website: openai.com
2. Log in!
3. Go to fine tuning: platform.openai.com/finetune
4. Follow the steps mentioned above and fine tune your own model

4.7 Integrating into the IDE

Integrating a fine-tuned OpenAI model into the IDE can be done through tools like Continue or custom extensions. Continue supports fine-tuned models natively—by adding the model's ID and API key to its configuration, developers can use the tuned assistant directly in the editor. This enables secure, org-specific completions and chat interactions. Alternatively, you can build a lightweight VS Code extension that sends the current file or prompt to your fine-tuned model via the OpenAI API and displays the response inline. This offers flexibility in UI/UX and workflow integration. Both approaches allow the model to be embedded into daily development tasks without changing tools.

5 Summary

In this workshop, we explored how to build AI coding assistants tailored for secure development.
We discussed the risks of insecure code generation, examined common vulnerabilities like SSRF, file uploads, and race conditions, and learned how to mitigate them through prompt engineering and fine-tuning.

Attendees practiced hands-on: crafting prompts, testing model responses, refining code, creating assistants, and fine-tuning custom models.
We concluded by integrating these models into development environments for real-world use.

6 Best Practices

- Always validate LLM outputs—never assume they're secure by default.
- Use clear, specific prompts, and include known constraints or risks.
- When possible, prefer fine-tuning over long system prompts to reduce latency and cost.
- Escape all special characters properly in '.jsonl' data (`\n`, `\"`, `\\`).
- Test models with adversarial prompts to uncover blind spots.
- Track API usage and set spend limits to avoid accidental overuse.
- Version datasets, prompts, and assistant configurations for reproducibility.

7 Q&A and Troubleshooting

- Q: My assistant still generates insecure code—why?
  A: Review prompt clarity, refine examples, and consider fine-tuning with better completions.
- Q: How can I integrate my model into an IDE?
  A: Use tools like Continue (via OpenAI model ID) or build a custom extension using the API.
- Q: What's the difference between assistants and fine-tuned models?
  A: Assistants rely on system prompts at runtime; fine-tuned models internalize patterns and respond faster and cheaper with consistent output.
- Q: Can I reuse fine-tuned models across projects?
  A: Yes, as long as they are accessible under your OpenAI account and API key.
- Q: Can fine-tuning fix hallucinations?
  A: It reduces them when the completions are well-curated and consistent, but does not eliminate them entirely.

References

[1] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, "Lost at C: A user study on the security implications of large language model code assistants," in *Proc. USENIX Security Symp.*, 2023.

[2] H. Pearce, et al., "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," *Commun. ACM*, vol. 68, no. 2, pp. 96-105, 2025.

[3] R. Khoury, A. R. Avila, J. Brunelle, and B. M. Camara, "How secure is code generated by ChatGPT?" in *CoRR*, vol. abs/2304.09655, 2023.

[4] MITRE, "CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition," *Common Weakness Enumeration*, https://cwe.mitre.org/data/definitions/367.html, accessed June 25, 2025.

[5] PortSwigger Academy. *Race conditions*. `https://portswigger.net/web-security/race-conditions`