GitHub Copilot Certification

Exam Preparation Guide (GH-300)

Complete Reference Handout

Cluster Reply GmbH

Internal Certification Course

February 2026 | Version 1.0

About This Handout

This document provides comprehensive coverage of all seven domains tested in the GitHub Copilot Certification Exam (GH-300). Use it to prepare for the exam and as a reference during your studies.

Contents

1 Exam Overview

Property	Details
Format	Multiple choice and scenario-based questions
Questions	60
Duration	120 minutes (2 hours)
Passing Score	Approximately 70% (42 correct answers)
Cost	$99 USD (varies by region)
Validity	2 years from passing date
Retake Policy	24 hours after first attempt

Table 1: Exam format at a glance

1.1 Domain Weights

Domain	Topic	Weight
Domain 1	Responsible AI	7%
Domain 2	Plans and Features	31%
Domain 3	Data Pipeline	15%
Domain 4	Prompt Engineering	9%
Domain 5	Developer Use Cases	14%
Domain 6	Testing	9%
Domain 7	Privacy and Exclusions	15%
	Total	**100%**

Table 2: Domain weights – Domain 2 is the largest at 31%

1.2 Target Audience

This certification is designed for:

- Software developers using GitHub Copilot
- DevOps engineers and administrators
- Project managers overseeing AI tool adoption
- Technical leads evaluating Copilot for teams

1.3 Prerequisites

No formal prerequisites, but recommended:

- Foundational understanding of GitHub
- Hands-on experience with GitHub Copilot
- Familiarity with at least one programming language
- Basic understanding of AI/ML concepts

2 Domain 1: Responsible AI (7%)

Microsoft's Responsible AI framework governs all AI products including GitHub Copilot. This domain tests your understanding of ethical AI principles and their practical application.

2.1 The Six Principles of Responsible AI

Memorize

Mnemonic: "FRPITA"

Fairness · **R**eliability & Safety · **P**rivacy & Security · **I**nclusiveness · **T**ransparency · **A**ccountability

2.1.1 1. Fairness

Definition: AI treats every user equitably, avoiding systematic bias.

Application to Copilot:

- Consistent code suggestions regardless of user background
- Equal quality across programming languages
- No discrimination based on coding style or region
- Bias detection and mitigation in suggestions

Key Concept

Copilot should amplify human capabilities without discrimination. A junior developer and senior architect should receive contextually appropriate suggestions of equal quality.

Exam Scenario

"Copilot provides better Python suggestions than Ruby suggestions despite similar context."
Answer: This is a **Fairness** concern.

2.1.2 2. Reliability and Safety

Definition: AI systems perform predictably and minimize risks.

Application to Copilot:

- Consistent behavior for similar inputs
- Rigorous testing against edge cases
- Resilient design against malicious inputs
- Adherence to secure coding standards

Key Concept

Reliability means **consistent** behavior, not **correct** behavior. Copilot can still suggest incorrect code, but it should fail gracefully and behave predictably.

Exam Scenario

"A security team wants assurance Copilot won't generate malware when prompted maliciously."
Answer: This relates to **Reliability and Safety**.

2.1.3 3. Privacy and Security

Definition: Protecting user data, code, and usage patterns.

Application to Copilot:

- Azure encryption for all data (transit and rest)
- Hardware Security Modules (HSMs) for key management
- GitHub-owned Azure tenants (no third-party access)
- Clear data retention policies

Memorize

Data Retention:
- Business/Enterprise: Prompts **NEVER** used for training
- User engagement data: Retained **2 years**
- Chat history: Retained **28 days**
- Prompts: Discarded after response generation

Exam Scenario

"A CISO asks how proprietary code is protected."
Answer: Azure encryption, HSMs, Business/Enterprise code **never used for training**.

2.1.4 4. Inclusiveness

Definition: AI should benefit everyone regardless of ability, location, or background.

Application to Copilot:

- Screen reader compatibility
- Full keyboard navigation
- Support for multiple natural languages in comments
- Adaptation to regional coding conventions

Exam Scenario

"A visually impaired developer asks if Copilot is accessible."
Answer: This relates to **Inclusiveness**.

2.1.5 5. Transparency

Definition: Revealing how AI models make decisions.

Application to Copilot:

- `/explain` command for understanding suggestions
- Debugging tools for tracing logic
- Usage dashboards (Enterprise)
- Audit trails for compliance

Exam Scenario

"A developer wants to understand why Copilot suggested a specific algorithm."
Answer: Use `/explain` command (**Transparency** principle).

2.1.6 6. Accountability

Definition: Clear ownership for AI outcomes.

Application to Copilot:

- Organizations must define roles and responsibilities
- Monitor system performance and usage
- Conduct regular audits
- Address harms promptly
- Microsoft maintains ongoing oversight

Critical

The **developer** is accountable for AI-generated code—not Copilot, not GitHub, not Microsoft. You review it, you commit it, you own it.

Exam Scenario

"Who is responsible when Copilot-suggested code causes a production bug?"
Answer: The **developer** who accepted and committed the code.

2.2 AI Limitations and Risks

Limitations:

- Training data may be outdated
- Limited context window (cannot see entire codebase)
- Cannot perform mathematical calculations reliably
- May suggest insecure coding patterns
- Bias inherited from training data
- No true understanding of intent (pattern matching only)

Mitigation Strategies:

- Always review and validate output
- Use code scanning tools (GitHub Advanced Security, Dependabot)
- Apply security best practices
- Test thoroughly before deployment
- Understand context window limitations
- Provide clear, specific prompts

Exam Tip

Expect 3–5 questions on Responsible AI principles. Questions are scenario-based—know the principles well enough to recognize them in context.

3 Domain 2: GitHub Copilot Plans and Features (31%)

This is the **largest** exam domain at 31%—approximately 18–19 questions. Master every detail of plan differences and features.

3.1 GitHub Copilot Plans Comparison

3.1.1 Copilot Free

Price: $0 | **Completions:** 2,000/month | **Premium Requests:** 50/month
Target: Developers trying Copilot
Key Limitations: No IP indemnity, no policy management, no audit logs, limited model access, not available if you have a Business/Enterprise seat.

3.1.2 Copilot Pro

Price: $10/month ($100/year) | **Completions:** Unlimited | **Premium Requests:** 300/month
Target: Individual developers
Key Features: Full Chat, multiple AI models, Coding Agent access, free for verified students/teachers/OSS maintainers.
Key Limitations: No IP indemnity, no organization controls, prompts may contribute to training (opt-out available).

3.1.3 Copilot Pro+

Price: $39/month | **Completions:** Unlimited | **Premium Requests:** 1,500/month
Target: Power users needing advanced models
Key Features: All Pro features, access to ALL available AI models, highest premium request allowance for individuals, priority access to new features.

3.1.4 Copilot Business

Price: $19/user/month | **Completions:** Unlimited | **Premium Requests:** 300/user/month
Target: Teams and organizations
Key Features: IP Indemnity, centralized license management, org-wide policy controls, audit logs, content exclusions, code **NEVER** used for training, Coding Agent access.
Key Limitations: No Knowledge Bases, no custom models.

3.1.5 Copilot Enterprise

Price: $39/user/month | **Completions:** Unlimited | **Premium Requests:** 1,000/user/month
Requirement: GitHub Enterprise Cloud subscription
Target: Large organizations
Key Features: All Business features, Knowledge Bases, custom models, Chat on GitHub.com, PR summaries, enhanced security controls, SSO & SCIM support.

3.2 Feature Comparison Table

Feature	Free	Pro/Pro+	Business	Enterprise
IP Indemnity	✗	✗	✓	✓
Policy Management	✗	✗	✓	✓
Knowledge Bases	✗	✗	✗	✓
Coding Agent	✗	✓	✓	✓
Audit Logs	✗	✗	✓	✓
Content Exclusions	✗	✗	✓	✓
Code Used for Training	Maybe	Maybe	**NEVER**	**NEVER**
Custom Models	✗	✗	✗	✓
PR Summaries	✗	✗	✗	✓

Table 3: Feature availability by plan

3.3 Premium Request Limits

Memorize

Plan	Monthly Limit	Overage Cost
Free	50	N/A
Pro	300	$0.04 each
Pro+	1,500	$0.04 each
Business	300/user	$0.04 each
Enterprise	1,000/user	$0.04 each

3.4 Interaction Methods

3.4.1 1. Inline Suggestions (Ghost Text)

Gray “ghost text” appears as you type, providing real-time, context-aware suggestions.

Memorize

Keyboard Shortcuts:

Action	Windows	Mac
Accept suggestion	`Tab`	`Tab`
Dismiss suggestion	`Esc`	`Esc`
Next suggestion	`Alt+]`	`Option+]`
Previous suggestion	`Alt+[`	`Option+[`

Supported IDEs: VS Code, Visual Studio, JetBrains (IntelliJ, PyCharm, WebStorm, Rider, GoLand, etc.), Neovim, Vim, Xcode, Azure Data Studio.

3.4.2 2. Copilot Chat

Conversational interface for complex interactions.

Memorize

Chat Shortcuts:

- **Open Chat Panel:** `Ctrl+Alt+I` (Win) / `Cmd+Option+I` (Mac)
- **Inline Chat:** `Ctrl+I` (Win) / `Cmd+I` (Mac)

Slash Commands:

Command	Purpose
`/explain`	Explain selected code
`/fix`	Suggest fixes for errors
`/tests`	Generate unit tests
`/doc`	Generate documentation
`/optimize`	Suggest performance improvements
`/clear`	Clear chat history

3.4.3 3. Copilot CLI

Terminal-based interaction via the GitHub CLI.

Commands:

- `gh copilot explain` – Explain a command or concept
- `gh copilot suggest` – Get command suggestions

Use Cases: Shell command help, Git operations, DevOps scripting, understanding unfamiliar CLI tools.

Installation: Requires GitHub CLI (`gh`) with Copilot extension.

3.5 Knowledge Bases (Enterprise Only)

Definition: Indexed collections of your organization's code that Copilot uses to provide personalized suggestions.

Can Store:

- Code snippets and patterns
- Best practices documentation
- Design patterns and API usage examples
- Team conventions

Benefits: Suggestions aligned with your codebase, consistent code style across teams, faster onboarding, reduced context switching.

Configuration: Create via GitHub.com organization settings; index specific repositories; configure refresh schedules; set access permissions.

Exam Tip

Know exact premium request limits, which features are in which plans, and that IP indemnity is Business/Enterprise **ONLY**.

4 Domain 3: How GitHub Copilot Works and Handles Data (15%)

This domain covers the technical architecture of Copilot's data pipeline—how your code becomes a suggestion.

4.1 The Four-Stage Data Pipeline

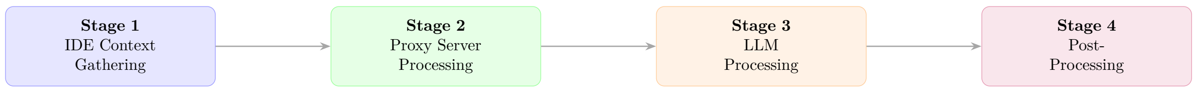

Figure 1: The four-stage Copilot data pipeline

4.1.1 Stage 1: IDE Context Gathering

What Copilot Collects:

- Current file content (primary context)
- Cursor position and surrounding code
- Open tabs (neighboring files)
- File paths and repository structure
- Repository URLs
- Chat history (for Chat interactions)

Key Concept

Context Priority:

1. Current file (highest priority)
2. Open tabs in editor
3. Recently edited files
4. File path semantics

Tip: Open related files to improve suggestion quality (e.g., models and interfaces).

4.1.2 Stage 2: Proxy Server Processing

Security Filters Applied:

- Authentication & authorization verification
- Rate limiting enforcement
- PII (Personally Identifiable Information) detection
- Toxic language screening
- Relevance checks
- Jailbreak attempt prevention

4.1.3 Stage 3: LLM Processing

Infrastructure: Hosted in GitHub-owned Azure tenants. Multiple models available (GPT-4, Claude, and others—varies by plan).

Processing: Prompt assembled from context → model generates completion → multiple candidates may be generated → ranking determines best suggestion.

4.1.4 Stage 4: Post-Processing

Quality Checks: Syntax validation, code quality assessment, security pattern detection.

Public Code Filter:

- Compares against indexed public repositories
- Threshold: **150+ characters** matching
- Whitespace stripped before comparison
- Matching suggestions blocked
- Approximately **1%** of suggestions match public code

4.2 Data Retention Policies

Data Type	Individual Plans	Business/Enterprise
Prompts	May be used for training (opt-out)	**NEVER** used; discarded after response
User engagement data	2 years	2 years
Chat history	28 days	28 days
Audit logs	N/A	Available

Table 4: Data retention by plan type

4.3 Public Code Filter Details

Memorize

Public Code Filter (Exam Favorite):

- **Threshold:** 150+ characters
- **Comparison:** Whitespace stripped before matching
- **Match Rate:** ∼1% of suggestions
- **Configuration:** Enable/disable per organization
- **Purpose:** Prevent accidental inclusion of copyrighted/GPL code

Why 150 characters? Short snippets are often generic (imports, loops); longer matches more likely indicate copied code. This balances usability with IP protection.

4.4 Context Window Limitations

- Copilot can only “see” a limited amount of code
- Cannot understand your entire codebase
- Suggestions may conflict with code in other files
- Recent changes in closed files not considered

Mitigation: Open relevant files in editor tabs, add descriptive comments, use clear function signatures, provide context in Chat prompts.

4.5 LLM Limitations

- **Outdated training data:** May not know newest APIs or frameworks
- **Pattern matching only:** Predicts text, doesn’t “understand” code
- **Hallucinations:** May invent non-existent APIs or functions
- **Math:** Cannot reliably perform calculations

Exam Tip

Know the 150-character threshold for the public code filter, the four pipeline stages, and that Business/Enterprise prompts are **NEVER** used for training.

5 Domain 4: Prompt Crafting and Prompt Engineering (9%)

5.1 Context Sources for Prompts

Memorize

Copilot builds prompts from **five sources** (priority order):

1. **Current file** – Primary context; content before/after cursor, language, imports
2. **Open tabs** – Neighboring files; open 1–3 related files for best results
3. **File paths** – Directory names provide semantic hints (e.g., `/src/auth/`)
4. **Comments** – Natural language descriptions; write intent *before* code
5. **Chat history** – Previous exchanges inform responses

5.2 Prompting Best Practices

1. **Be Specific**
   Bad: "make a function"
   Good: "Create a TypeScript function that validates email addresses using regex, returns boolean, handles edge cases like plus addressing"
2. **Provide Context**
   Include what you're accomplishing, constraints, expected I/O, error handling needs.
3. **Break Down Tasks**
   Instead of "Build a user authentication system," ask separately for the user model, password hashing, login endpoint, and session management.
4. **Iterate and Refine**
   Add constraints: "but without external libraries." Clarify: "I meant for browser, not Node.js."

5.3 Prompting Techniques

Technique	Examples	Best For
Zero-shot	0	Simple, common tasks; standard library operations
One-shot	1	Establishing a format; template following
Few-shot	2–5	Complex transformations; consistency-critical tasks
Chain-of-thought	N/A	Complex reasoning; debugging; architecture decisions

Table 5: Prompting techniques comparison

5.3.1 Zero-Shot Prompting

No examples provided—rely on Copilot's training data.

```
// Validate email address
function validateEmail(email) {
  // Copilot generates implementation
}
```

Pros: Fast, no setup. **Cons:** May not match your specific style.

5.3.2 One-Shot Prompting

Single example guides the format/style.

```
// add(1, 2) returns 3
// subtract(a, b)
function subtract(a, b) {
  return a - b;  // Copilot follows the pattern
}
```

Pros: More predictable. **Cons:** Single example may not cover edge cases.

5.3.3 Few-Shot Prompting

Multiple examples (2–5) establish a clear pattern.

```
// "hello" -> "HELLO"
// "World" -> "WORLD"
// "test"  -> ?
// Result: "TEST"
```

Pros: Most reliable for pattern matching. **Cons:** Uses more context tokens.

5.3.4 Chain-of-Thought Prompting

Break complex problems into reasoning steps using trigger phrases:

- "Think step by step"
- "Let's break this down"
- "Explain your reasoning"
- "Walk me through the logic"

Key Concept

The Trade-Off:
More examples = more predictable output, but also more context tokens used.
Balance: Simple tasks → zero-shot; format guidance → one-shot; complex patterns → few-shot; reasoning → chain-of-thought.

Exam Tip

Know the difference between zero-shot, one-shot, and few-shot prompting. Understand when to use chain-of-thought for complex reasoning tasks.

6 Domain 5: Developer Use Cases for AI (14%)

6.1 Eight Key Productivity Use Cases

Memorize

1. Learning new languages and frameworks
2. Language translation (code conversion)
3. Context switching
4. Writing documentation (`/doc`)
5. Sample data generation
6. Legacy code modernization
7. Debugging code (`/fix`)
8. Code refactoring

6.1.1 1. Learning New Languages and Frameworks

Copilot explains unfamiliar syntax, suggests idiomatic patterns, provides language-specific best practices, and translates concepts from known languages.

Tip: Use `/explain` on existing code when learning.

6.1.2 2. Language Translation

Converts code between programming languages, handles syntax differences, adapts naming conventions, and translates idioms appropriately.

6.1.3 3. Context Switching

Maintains mental model across files, remembers conversation context, reduces cognitive load, and bridges between related components.

Tip: Keep related files open when switching contexts.

6.1.4 4. Writing Documentation

Generates docstrings/JSDoc, README content, inline comments, and API documentation via the `/doc` command.

Supported formats: JSDoc (JS/TS), Docstrings (Python), XML comments (C#), JavaDoc (Java).

6.1.5 5. Sample Data Generation

Creates realistic test fixtures, mock data, and seed data for databases. Respects locale conventions.

6.1.6 6. Legacy Code Modernization

Refactors old patterns to modern standards: callbacks to async/await, deprecated API updates, jQuery to vanilla JS, etc.

6.1.7 7. Debugging Code

Explains error messages, identifies root causes, suggests fixes, and traces logic flow via the `/fix` command.

6.1.8 8. Code Refactoring

Improves code structure, reduces duplication, enhances readability, and applies design patterns.

6.2 SDLC Integration

Phase	Copilot Assistance
Planning	User stories, acceptance criteria, technical specifications
Coding	Inline suggestions, chat assistance, code completion
Testing	Test generation (`/tests`), edge cases, mock data
Review	PR summaries (Enterprise), explanations, security analysis
Deployment	Script generation, CI/CD pipelines, infrastructure code

Table 6: Copilot across the Software Development Lifecycle

6.3 Productivity API (Enterprise)

Purpose: Measure Copilot's impact on development efficiency.

Metrics: Acceptance rates, time saved, language-specific usage, team-wide statistics, trends over time.

Use Cases: Justify investment, identify training needs, optimize workflows.

Exam Tip

Know all eight use cases and be able to identify which applies to a given scenario. Understand SDLC integration points.

7 Domain 6: Testing with GitHub Copilot (9%)

7.1 Test Generation Capabilities

7.1.1 Unit Tests

Command: `/tests`

Process: Select code → use `/tests` → Copilot generates test structure → review and customize → run tests.

Generated content: Test file structure, import statements, describe/it blocks, multiple test cases, assertions.

7.1.2 Integration Tests

Copilot understands HTTP methods, request/response patterns, authentication requirements, and error handling scenarios. Generates setup/teardown, API calls, response validation, and error case handling.

7.1.3 Edge Case Tests

Copilot identifies: null/undefined inputs, empty arrays/strings, max/min values, type coercion issues, concurrent access, and resource exhaustion.

7.1.4 Assertion Generation

Generates framework-appropriate matchers, happy path and error case assertions, and type checking assertions.

7.2 Key Testing Features

1. **Learns from Existing Tests**
   Copilot analyzes your existing tests and matches framework, assertion patterns, naming conventions, setup/teardown structure, and mocking patterns.
   Tip: Open existing test files to help Copilot understand your team's testing style.
2. **Edge Case Suggestion**
   Identifies boundary values (0, −1, MAX_INT), error conditions, null handling, empty collections, invalid input types.
3. **Boilerplate Generation**
   Creates complete test file structure including imports, organization, hooks, mock configuration, and utilities.
4. **Test Improvement**
   Analyzes existing tests and suggests missing coverage, additional assertions, better organization, and performance improvements.

7.3 SKU Privacy Considerations

Plan Type	Privacy
Individual (Free/Pro/Pro+)	Test code may contribute to training; opt-out available
Business/Enterprise	Test code **NEVER** used for training; full privacy controls; audit logging

7.4 Testing Best Practices with Copilot

1. Start with clear function signatures—well-typed code gets better test suggestions
2. Open existing tests—Copilot learns your testing patterns
3. Review generated tests—don't blindly trust; validate logic
4. Add edge cases manually—Copilot may miss domain-specific cases
5. Maintain test quality—generated tests are starting points, not finished products

Exam Tip

Know that Copilot learns from existing tests to maintain consistency. Understand the `/tests` command and what it generates.

8 Domain 7: Privacy Fundamentals and Context Exclusions (15%)

8.1 Content Exclusions

Purpose: Prevent specific code from being sent to Copilot servers—proprietary algorithms, security-sensitive code, regulated data (HIPAA, PCI, GDPR), trade secrets, API keys/credentials.

8.1.1 Configuration Levels

Level	Configuration Location
Repository	`.github/copilot-exclusions.yml`
Organization	GitHub.com → Organization Settings → Copilot

Example `.github/copilot-exclusions.yml`**:**

```
exclusions:
  - "**/.env"
  - "**/.env.*"
  - "**/secrets/**"
  - "src/proprietary/**"
  - "*.key"
  - "*.pem"
  - "config/credentials.yml"
```

8.1.2 Effects of Exclusions

When content is excluded:

1. File content **NOT** sent to Copilot servers
2. No suggestions generated **FROM** excluded files
3. No suggestions generated **WITHIN** excluded files
4. Applies to **BOTH** Chat and inline suggestions
5. Affects **all users** in the organization

Critical

Critical Limitation (Exam Favorite):
Exclusions do **NOT** prevent Copilot from suggesting *similar* code that exists in its training data. You exclude your proprietary sorting algorithm—Copilot won't see *your* code, but might still suggest a similar algorithm it learned from public repositories. This is expected behavior, not a security failure.

8.2 Duplication Detector Filter (Safeguard)

Memorize

- **Threshold:** 150+ characters
- **Comparison:** Whitespace stripped before matching
- **Match Rate:** ∼1% of suggestions
- **Configuration:** Per organization (enable/disable)
- **Process:** Generate → compare → strip whitespace → 150+ match → block

8.3 IP Indemnity (Business & Enterprise Only)

What it means: GitHub legally defends against IP infringement claims covering **unmodified** Copilot suggestions.

Coverage: Third-party IP claims, copyright infringement, patent disputes.

Critical

Indemnity covers **unmodified suggestions only**. If you substantially modify the suggestion, standard IP rules apply.

8.4 GitHub.com Organization Settings

Administrators can configure:

1. **Duplication detection** – Enable/disable public code matching filter

2. **Prompt & suggestion collection** – Enable for model improvement or disable (Business/Enterprise default: disabled)
3. **Copilot access** – Enable/disable for organization; control team-level access
4. **Content exclusions** – Organization-wide exclusion patterns

8.5 Troubleshooting

Problem	**Cause**	**Solution**
No suggestions for some files	Content exclusion active	Check `.github/copilot-exclusions.yml` and org settings
Exclusions not applying	Sync delay or YAML error	Validate YAML; wait 5 min; restart IDE
Suggestions completely absent	Rate limit or network	Check GitHub status; verify connectivity
Poor suggestion quality	Insufficient context	Open related files; add comments; add types

Table 7: Common troubleshooting scenarios

8.5.1 Systematic Troubleshooting Checklist

1. **File type supported?** – Check language/extension; binary files never get suggestions
2. **Extension enabled?** – Verify Copilot installed; check status bar icon
3. **Content exclusions?** – Review org and repo settings
4. **Network connectivity?** – Copilot requires internet; check firewall
5. **Rate limit status?** – Free plan: 2,000 completions/month
6. **Authentication?** – Verify GitHub account connected; re-authenticate if needed

8.5.2 Why Exclusions May Not Apply

1. **YAML syntax error** – Invalid indentation, missing quotes, incorrect patterns
2. **Sync delay** – Changes take up to 5 minutes; restart IDE to force refresh
3. **Pattern mismatch** – File doesn't match; case sensitivity; path separator differences
4. **Scope mismatch** – Repo vs. org exclusion; wrong repository targeted

8.6 Ownership of Copilot Outputs

Key Concept

Key Principle: GitHub does **NOT** claim ownership of suggestions. **You** own the code you accept and commit.
Responsibility Chain: Copilot suggests → **you** review and accept → **you** commit → **you** are responsible.

Exam Tip

Know content exclusion configuration, the 150-character duplication threshold, IP indemnity limitations, and systematic troubleshooting steps.

9 Quick Reference Tables

9.1 Keyboard Shortcuts

Action	Windows	Mac
Accept suggestion	`Tab`	`Tab`
Dismiss suggestion	`Esc`	`Esc`
Next suggestion	`Alt+]`	`Option+]`
Previous suggestion	`Alt+[`	`Option+[`
Open Chat Panel	`Ctrl+Alt+I`	`Cmd+Option+I`
Inline Chat	`Ctrl+I`	`Cmd+I`

9.2 Slash Commands

Command	Purpose
`/explain`	Explain selected code
`/fix`	Suggest fixes for errors
`/tests`	Generate unit tests
`/doc`	Generate documentation
`/optimize`	Suggest performance improvements
`/clear`	Clear chat history

9.3 Plan Pricing and Premium Requests

Plan	Price	Premium Req./Month	Overage
Free	$0	50	N/A
Pro	$10/mo	300	$0.04
Pro+	$39/mo	1,500	$0.04
Business	$19/user/mo	300/user	$0.04
Enterprise	$39/user/mo	1,000/user	$0.04

9.4 Data Retention Summary

Data Type	Retention
User engagement data	2 years
Chat history	28 days
Prompts (Individual)	May be used for training
Prompts (Business/Ent.)	Discarded after response

9.5 Responsible AI Principles

#	Principle	Keyword
1	Fairness	Equitable
2	Reliability & Safety	Predictable
3	Privacy & Security	Protected
4	Inclusiveness	Everyone
5	Transparency	Explainable
6	Accountability	Ownership

9.6 Context Sources (Priority Order)

Priority	Source
1 (highest)	Current file
2	Open tabs
3	File paths
4	Comments
5	Chat history

9.7 Duplication Detection

Setting	Value
Threshold	150+ characters
Whitespace	Stripped before comparison
Match Rate	~1% of suggestions
Configurable	Per organization

10 Exam Preparation Checklist

10.1 Domain 1: Responsible AI (7%)

- ☐ Memorize all 6 principles
- ☐ Know practical applications of each
- ☐ Understand AI limitations
- ☐ Know mitigation strategies
- ☐ Remember: Developer is accountable

10.2 Domain 2: Plans and Features (31%)

- ☐ Memorize all 5 plans and prices
- ☐ Know premium request limits exactly
- ☐ Understand feature differences
- ☐ Know IP indemnity is Business/Enterprise only
- ☐ Know Knowledge Bases are Enterprise only
- ☐ Memorize keyboard shortcuts
- ☐ Know all slash commands
- ☐ Understand CLI commands

10.3 Domain 3: Data Pipeline (15%)

- ☐ Know 4 pipeline stages
- ☐ Understand context gathering
- ☐ Know proxy filters
- ☐ Remember 150-char duplication threshold
- ☐ Know data retention periods
- ☐ Understand Business/Enterprise never trains

10.4 Domain 4: Prompt Engineering (9%)

- ☐ Know 5 context sources
- ☐ Understand 4 best practices
- ☐ Know zero/one/few-shot differences
- ☐ Understand chain-of-thought prompting
- ☐ Know when to use each technique

10.5 Domain 5: Developer Use Cases (14%)

- ☐ Memorize all 8 use cases
- ☐ Know SDLC integration points
- ☐ Understand Productivity API
- ☐ Know Copilot limitations

10.6 Domain 6: Testing (9%)

- ☐ Know `/tests` command
- ☐ Understand test types generated
- ☐ Know Copilot learns from existing tests
- ☐ Understand SKU privacy differences
- ☐ Know editor config options

10.7 Domain 7: Privacy and Exclusions (15%)

- ☐ Know exclusion configuration locations
- ☐ Understand exclusion effects
- ☐ Remember exclusion limitation
- ☐ Know duplication detection details
- ☐ Understand IP indemnity scope
- ☐ Master troubleshooting steps

10.8 Exam Day Preparation

- ☐ Get good sleep the night before
- ☐ Arrive 15 minutes early
- ☐ Bring valid ID
- ☐ Review quick reference tables
- ☐ Remember: 2 minutes per question average
- ☐ Flag difficult questions and return

11 Study Resources

11.1 Official Resources

Microsoft Learn:

- GitHub Copilot Fundamentals (Part 1 & 2)
- Responsible AI with GitHub Copilot
- Practice Assessment

GitHub Documentation:

- `https://docs.github.com/copilot` – Main documentation
- Subscription plans, data handling, features by plan

GitHub Skills:

- `https://skills.github.com` – Hands-on exercises

11.2 Practice Recommendations

1. Use Copilot daily for at least 2 weeks before the exam
2. Try all interaction methods (inline, chat, CLI)
3. Experiment with different prompting techniques
4. Test content exclusions in a repository
5. Review audit logs (if on Business/Enterprise)

11.3 Exam Registration

Property	**Details**
Website	`https://learn.microsoft.com/credentials`
Cost	$99 USD
Duration	120 minutes
Questions	60
Format	Multiple choice, scenario-based
Proctoring	Online or test center

12 Final Notes

12.1 Key Exam Strategies

1. **Read questions carefully** – Many questions have subtle differences in answer choices
2. **Eliminate wrong answers** – Often 2 answers are clearly wrong; focus on the remaining 2
3. **Watch for absolutes** – “Always,” “never,” “only” often indicate wrong answers
4. **Scenario questions** – Identify what’s being asked *before* looking at answers
5. **Time management** – 2 minutes per question; flag and move on if stuck
6. **Trust your preparation** – First instinct is often correct; don’t second-guess

12.2 Common Exam Traps

1. Confusing IP indemnity availability (Business/Enterprise **ONLY**)
2. Wrong premium request limits
3. Confusing exclusion effects vs. limitations
4. Wrong keyboard shortcuts

5. Mixing up plan features
6. Forgetting developer accountability

Good luck on your GitHub Copilot Certification Exam!

You've studied all 7 domains. You know the exact numbers. You understand the principles, not just the facts. You can apply knowledge to scenarios.