GitHub Copilot → Agentic Systems — Cheat Sheet

The whole course on one page. Specifics reflect **mid-2026**; verify at docs.github.com/copilot.

Completions	Ask	Edit	Agent mode	Cloud agent	Multi-agent
predict a line	understand	change what you know	do what you describe	async → draft PR	orchestrate a fleet

YOU DRIVE — hand off more as trust grows — AGENTS DRIVE

The agent loop

Reason → Act → Observe → repeat. A chatbot answers; an agent *tries, sees what happened, adjusts.*

Agent = Model + Tools + Memory + Planning. No tools → it's just a chatbot.

Autonomy levels: assistant → co-pilot → supervised → autonomous → orchestrator. *Match autonomy to blast radius.*

Chat modes & agents

Ask	Q&A, no file changes — scope it here first.
Edit	Change files you name; shows a diff.
Agent	Describe a task; it picks files, runs, iterates.

Agent mode = synchronous, in your editor, you watch.
Cloud agent = async, sandboxed Actions, returns a **draft PR**.

Models

Chat model and **completions model** are set **separately** (completions: "Change Completions Model" command).

Auto routes by task + availability; hover a reply to see which ran.

BYOK covers chat, tools, MCP, utility tasks. **NOT completions, semantic search, or embeddings (need sign-in).** Billed by your provider, not Copilot quota.

Which model when

Everyday	Auto
Hard reasoning	Top frontier model + high effort
Fast/simple	A fast/cheap model
Huge codebase	1M-context model
Agentic task	Tool-calling model
Private code	Local model (Ollama/Foundry)
Offline	Local + BYOK (no completions)
Out of budget	0×–1× model or local

Build your own agent (4 ways)

1 Agent Skill — `SKILL.md` teaches a procedure (lowest effort).
2 Custom Agent — a specialized cloud agent.
3 Extension — GitHub App: *skillset* (≤5 endpoints, Copilot does the AI) vs *agent* (you control prompt/model/flow).
4 Copilot SDK — embed the same runtime as Copilot CLI; BYOK optional.

MCP in 30 seconds

Roles: Host (the app) → Client (1 per server) → Server (a domain). The server never talks to the model directly.

Server primitives: Tools (act), Resources (read-only), Prompts (templates). *Resources query, tools act, prompts standardize.*

Client primitives: Sampling, Roots, Elicitation (human-in-the-loop).

Transports: `stdio` (local) vs Streamable HTTP (remote, OAuth 2.1; replaced SSE).

Config — same shape almost everywhere:

```
{ "mcpServers": {
  "filesystem": { "command": "npx",
    "args": ["-y",
      "@modelcontextprotocol/server-filesystem","."] }
}}
```

⚠ **VS Code uses "servers", NOT "mcpServers".**

Local LLMs

Ollama: `ollama pull llama3.1:8b` → `localhost:11434` (OpenAI-compatible at `/v1`).

Foundry Local: `winget install Microsoft.FoundryLocal`; ONNX Runtime, CPU/GPU/**NPU**, OpenAI-compatible.

VirtualBox = CPU only (no GPU passthrough); needs **AVX2**; use small Q4 models (~3–8B).

Host→VM: set `OLLAMA_HOST=0.0.0.0` + host-only adapter (no auth — trusted nets only).

Agent mode needs a **tool-calling** model (Llama 3.1+, Qwen2.5-Coder, Phi-4).

Guardrails (always)

- **Human gate** on irreversible actions (deploy, delete, spend).
- **Least-privilege** tools; **sandbox** risky execution.
- **Trace** every step; keep an audit trail.
- Treat external content as **untrusted data, not commands** (prompt injection). Check against OWASP ASI.

Pipelines & ops

Two pipelines: A = agents help ship your code (issue→plan→code→review→test→deploy). B = your AI product's request path (input→context→model+tools→output, all traced).

Evals replace exact-match tests — criteria-scored, gate the release.

Flywheel: prod failure → add an eval → fix prompt/tool → eval passes → redeploy. *Failures become permanent tests.*

Beyond Copilot

Four families: IDE extensions (Copilot, Cline, Continue), AI-native IDEs (Cursor, Windsurf, Zed), terminal agents (Claude Code, Codex, Aider), cloud platforms (Devin, Jules).

They **interoperate over MCP**; your MCP server + local model transfer across them.

Learn the standards, not the buttons.