Guide encyclopédique des IA en ligne de commande

Panorama exhaustif des outils CLI propulsés par l'IA générative

Thierry Gayet — linuxmaine.org

Mai 2026

Contents

1 Avant-propos **11**
- 1.1 Comment lire ce guide . . . 11
- 1.2 Mise en garde sur les versions . . . 11
- 1.3 Conventions typographiques . . . 11
- 1.4 Catégories couvertes . . . 11

2 Partie I — CLI IA propriétaires majeurs **13**
- 2.1 1.1 Claude Code (Anthropic) . . . 13
  - 2.1.1 Description . . . 13
  - 2.1.2 Caractéristiques . . . 13
  - 2.1.3 Installation . . . 13
  - 2.1.4 Refcard des commandes . . . 14
  - 2.1.5 Intégration IDE . . . 14
  - 2.1.6 Modèle économique . . . 14
- 2.2 1.2 OpenAI Codex CLI . . . 15
  - 2.2.1 Description . . . 15
  - 2.2.2 Caractéristiques . . . 15
  - 2.2.3 Installation . . . 15
  - 2.2.4 Refcard . . . 15
  - 2.2.5 Intégration IDE . . . 16
  - 2.2.6 Tarification . . . 16
- 2.3 1.3 Google Gemini CLI . . . 17
  - 2.3.1 Description . . . 17
  - 2.3.2 Caractéristiques . . . 17
  - 2.3.3 Installation . . . 17
  - 2.3.4 Refcard . . . 17
  - 2.3.5 Intégration IDE . . . 18
  - 2.3.6 Tarification . . . 18
- 2.4 1.4 GitHub Copilot CLI . . . 19
  - 2.4.1 Description . . . 19
  - 2.4.2 Caractéristiques . . . 19
  - 2.4.3 Installation . . . 19
  - 2.4.4 Refcard . . . 19
  - 2.4.5 Intégration IDE . . . 20
  - 2.4.6 Tarification . . . 20
- 2.5 1.5 Cursor CLI / Cursor Agent . . . 21
  - 2.5.1 Description . . . 21
  - 2.5.2 Caractéristiques . . . 21
  - 2.5.3 Installation . . . 21
  - 2.5.4 Refcard . . . 21
  - 2.5.5 Intégration IDE . . . 21
  - 2.5.6 Tarification . . . 21
- 2.6 1.6 Sourcegraph Cody CLI . . . 22
  - 2.6.1 Description . . . 22

2.6.2 Caractéristiques . . . 22
2.6.3 Installation . . . 22
2.6.4 Refcard . . . 22
2.6.5 Intégration IDE . . . 22
2.6.6 Tarification . . . 22
2.7 1.7 Amazon Q Developer CLI . . . 23
2.7.1 Description . . . 23
2.7.2 Caractéristiques . . . 23
2.7.3 Installation . . . 23
2.7.4 Refcard . . . 23
2.7.5 Intégration IDE . . . 23
2.7.6 Tarification . . . 24
2.8 1.8 Warp Terminal AI . . . 25
2.8.1 Description . . . 25
2.8.2 Caractéristiques . . . 25
2.8.3 Installation . . . 25
2.8.4 Refcard . . . 25
2.8.5 Intégration IDE . . . 25
2.8.6 Tarification . . . 25
2.9 1.9 Tabnine CLI . . . 26
2.9.1 Description . . . 26
2.9.2 Caractéristiques . . . 26
2.9.3 Installation . . . 26
2.9.4 Intégration IDE . . . 26
2.9.5 Tarification . . . 26
2.10 1.10 Autres CLI propriétaires notables . . . 27

3 Partie II — Agents de codage open-source 28
3.1 2.1 Aider . . . 28
3.1.1 Description . . . 28
3.1.2 Caractéristiques . . . 28
3.1.3 Installation . . . 28
3.1.4 Refcard . . . 29
3.1.5 Intégration IDE . . . 29
3.1.6 Tarification . . . 29
3.2 2.2 Plandex . . . 30
3.2.1 Description . . . 30
3.2.2 Caractéristiques . . . 30
3.2.3 Installation . . . 30
3.2.4 Refcard . . . 30
3.2.5 Intégration IDE . . . 31
3.2.6 Tarification . . . 31
3.3 2.3 Goose (Block / Square) . . . 32
3.3.1 Description . . . 32
3.3.2 Caractéristiques . . . 32
3.3.3 Installation . . . 32
3.3.4 Refcard . . . 32
3.3.5 Intégration IDE . . . 32
3.3.6 Tarification . . . 33
3.4 2.4 OpenHands (ex-OpenDevin) . . . 34
3.4.1 Description . . . 34
3.4.2 Caractéristiques . . . 34
3.4.3 Installation . . . 34
3.4.4 Refcard . . . 34
3.4.5 Intégration IDE . . . 35
3.4.6 Tarification . . . 35
3.5 2.5 Cline . . . 36

3.5.1 Description . . . 36
3.5.2 Caractéristiques . . . 36
3.5.3 Installation . . . 36
3.5.4 Refcard (CLI bêta) . . . 36
3.5.5 Intégration IDE . . . 36
3.5.6 Tarification . . . 36
3.6 2.6 Continue . . . 37
3.6.1 Description . . . 37
3.6.2 Caractéristiques . . . 37
3.6.3 Installation . . . 37
3.6.4 Refcard CLI . . . 37
3.6.5 Intégration IDE . . . 37
3.6.6 Tarification . . . 37
3.7 2.7 Open Interpreter . . . 38
3.7.1 Description . . . 38
3.7.2 Caractéristiques . . . 38
3.7.3 Installation . . . 38
3.7.4 Refcard . . . 38
3.7.5 Intégration IDE . . . 38
3.7.6 Tarification . . . 38
3.8 2.8 gpt-engineer . . . 39
3.8.1 Description . . . 39
3.8.2 Caractéristiques . . . 39
3.8.3 Installation . . . 39
3.8.4 Refcard . . . 39
3.8.5 Intégration IDE . . . 39
3.8.6 Tarification . . . 39
3.9 2.9 gptme . . . 40
3.9.1 Description . . . 40
3.9.2 Caractéristiques . . . 40
3.9.3 Installation . . . 40
3.9.4 Refcard . . . 40
3.9.5 Intégration IDE . . . 40
3.9.6 Tarification . . . 40
3.10 2.10 SWE-agent . . . 41
3.10.1 Description . . . 41
3.10.2 Caractéristiques . . . 41
3.10.3 Installation . . . 41
3.10.4 Refcard . . . 41
3.10.5 Intégration IDE . . . 41
3.10.6 Tarification . . . 41
3.11 2.11 Mentat . . . 42
3.11.1 Description . . . 42
3.11.2 Caractéristiques . . . 42
3.11.3 Installation . . . 42
3.11.4 Refcard . . . 42
3.11.5 Intégration IDE . . . 42
3.11.6 Tarification . . . 42
3.12 2.12 Smol Developer . . . 43
3.12.1 Description . . . 43
3.12.2 Caractéristiques . . . 43
3.12.3 Installation . . . 43
3.12.4 Refcard . . . 43
3.12.5 Tarification . . . 43
3.13 2.13 Charm Crush . . . 44
3.13.1 Description . . . 44

3.13.2 Caractéristiques . . . 44
3.13.3 Installation . . . 44
3.13.4 Refcard . . . 44
3.13.5 Intégration IDE . . . 44
3.13.6 Tarification . . . 45
3.14 2.14 Forge / Code Forge . . . 46
3.14.1 Description . . . 46
3.14.2 Caractéristiques . . . 46
3.14.3 Installation . . . 46
3.14.4 Refcard . . . 46
3.14.5 Tarification . . . 46
3.15 2.15 Roo Code . . . 47
3.15.1 Description . . . 47
3.15.2 Caractéristiques . . . 47
3.15.3 Installation . . . 47
3.15.4 Tarification . . . 47
3.16 2.16 Autres mentions notables . . . 48

4 Partie III — CLI LLM génériques 49
4.1 3.1 llm (Simon Willison) . . . 49
4.1.1 Description . . . 49
4.1.2 Caractéristiques . . . 49
4.1.3 Installation . . . 49
4.1.4 Refcard . . . 50
4.1.5 Backends . . . 50
4.1.6 Intégration IDE/shell . . . 50
4.1.7 Cas d'usage . . . 50
4.2 3.2 ShellGPT (sgpt) . . . 51
4.2.1 Description . . . 51
4.2.2 Caractéristiques . . . 51
4.2.3 Installation . . . 51
4.2.4 Refcard . . . 51
4.2.5 Backend Ollama . . . 51
4.2.6 Intégration shell . . . 51
4.3 3.3 aichat . . . 52
4.3.1 Description . . . 52
4.3.2 Caractéristiques . . . 52
4.3.3 Installation . . . 52
4.3.4 Refcard . . . 52
4.3.5 Backends . . . 52
4.3.6 Intégrations . . . 52
4.4 3.4 mods (Charmbracelet) . . . 53
4.4.1 Description . . . 53
4.4.2 Caractéristiques . . . 53
4.4.3 Installation . . . 53
4.4.4 Refcard . . . 53
4.4.5 Cas d'usage emblématiques . . . 54
4.5 3.5 tgpt . . . 55
4.5.1 Description . . . 55
4.5.2 Caractéristiques . . . 55
4.5.3 Installation . . . 55
4.5.4 Refcard . . . 55
4.6 3.6 fabric . . . 56
4.6.1 Description . . . 56
4.6.2 Caractéristiques . . . 56
4.6.3 Installation . . . 56
4.6.4 Refcard . . . 56

4.6.5 Cas d'usage . . . 56
4.7 3.7 gptscript . . . 57
4.7.1 Description . . . 57
4.7.2 Caractéristiques . . . 57
4.7.3 Installation . . . 57
4.7.4 Refcard . . . 57
4.8 3.8 chatgpt-cli (kardolus) . . . 58
4.8.1 Description . . . 58
4.8.2 Installation . . . 58
4.8.3 Refcard . . . 58
4.9 3.9 gpt-cli (kharvd) . . . 59
4.9.1 Description . . . 59
4.9.2 Installation . . . 59
4.9.3 Refcard . . . 59
4.10 3.10 smartcat (sc) . . . 60
4.10.1 Description . . . 60
4.10.2 Installation . . . 60
4.10.3 Refcard . . . 60
4.10.4 Intégration éditeur . . . 60
4.11 3.11 chatblade . . . 61
4.11.1 Description . . . 61
4.11.2 Installation . . . 61
4.11.3 Refcard . . . 61
4.12 3.12 gorilla-cli . . . 62
4.12.1 Description . . . 62
4.12.2 Installation . . . 62
4.12.3 Refcard . . . 62
4.13 3.13 shai (OVHcloud) . . . 63
4.13.1 Description . . . 63
4.13.2 Installation . . . 63
4.13.3 Refcard . . . 63
4.13.4 Backends . . . 63
4.14 3.14 wrangler ai (Cloudflare) . . . 64
4.14.1 Description . . . 64
4.14.2 Installation . . . 64
4.14.3 Refcard . . . 64
4.15 3.15 Autres clients CLI génériques . . . 65
4.15.1 Synthèse comparative — CLI génériques . . . 65

5 Partie IV — Runners de modèles LLM locaux 66
5.1 4.1 Ollama . . . 66
5.1.1 Description . . . 66
5.1.2 Caractéristiques . . . 66
5.1.3 Installation . . . 66
5.1.4 Refcard . . . 67
5.1.5 API OpenAI-compatible . . . 67
5.1.6 Intégration IDE . . . 67
5.1.7 Performance . . . 67
5.2 4.2 llama.cpp . . . 69
5.2.1 Description . . . 69
5.2.2 Caractéristiques . . . 69
5.2.3 Installation . . . 69
5.2.4 Refcard . . . 69
5.2.5 Intégration IDE . . . 70
5.2.6 Performance . . . 70
5.3 4.3 LM Studio CLI . . . 71
5.3.1 Description . . . 71

5.3.2 Caractéristiques . . . 71
5.3.3 Installation . . . 71
5.3.4 Refcard . . . 71
5.3.5 Intégration IDE . . . 71
5.4 4.4 llamafile (Mozilla Builders) . . . 72
5.4.1 Description . . . 72
5.4.2 Caractéristiques . . . 72
5.4.3 Installation . . . 72
5.4.4 Refcard . . . 72
5.4.5 Cas d'usage . . . 72
5.5 4.5 vLLM . . . 73
5.5.1 Description . . . 73
5.5.2 Caractéristiques . . . 73
5.5.3 Installation . . . 73
5.5.4 Refcard . . . 73
5.5.5 Intégration IDE . . . 73
5.5.6 Cas d'usage . . . 74
5.6 4.6 GPT4All . . . 75
5.6.1 Description . . . 75
5.6.2 Caractéristiques . . . 75
5.6.3 Installation . . . 75
5.6.4 Refcard . . . 75
5.6.5 Cas d'usage . . . 75
5.7 4.7 LocalAI . . . 76
5.7.1 Description . . . 76
5.7.2 Caractéristiques . . . 76
5.7.3 Installation . . . 76
5.7.4 Refcard . . . 76
5.7.5 Intégration IDE . . . 76
5.8 4.8 Jan . . . 77
5.8.1 Description . . . 77
5.8.2 Caractéristiques . . . 77
5.8.3 Installation . . . 77
5.8.4 Refcard CLI Cortex . . . 77
5.9 4.9 RamaLama . . . 78
5.9.1 Description . . . 78
5.9.2 Caractéristiques . . . 78
5.9.3 Installation . . . 78
5.9.4 Refcard . . . 78
5.9.5 Cas d'usage . . . 78
5.10 4.10 KoboldCpp . . . 79
5.10.1 Description . . . 79
5.10.2 Caractéristiques . . . 79
5.10.3 Installation . . . 79
5.10.4 Refcard . . . 79
5.11 4.11 text-generation-webui (Oobabooga) . . . 80
5.11.1 Description . . . 80
5.11.2 Installation . . . 80
5.11.3 Refcard . . . 80
5.12 4.12 mlx-lm (Apple) . . . 81
5.12.1 Description . . . 81
5.12.2 Installation . . . 81
5.12.3 Refcard . . . 81
5.13 4.13 Outils audio / image en CLI (bonus IA générative) . . . 82
5.13.1 4.13.1 whisper.cpp . . . 82
5.13.2 4.13.2 piper . . . 82

5.13.3 4.13.3 stable-diffusion.cpp . . . 82
5.13.4 4.13.4 Coqui TTS . . . 82
5.14 4.14 Synthèse — Runners locaux . . . 84

6 Partie V — Outils CLI spécialisés 85
6.1 5.1 aicommits . . . 85
6.1.1 Description . . . 85
6.1.2 Caractéristiques . . . 85
6.1.3 Installation . . . 85
6.1.4 Refcard . . . 85
6.2 5.2 opencommit (oco) . . . 87
6.2.1 Description . . . 87
6.2.2 Installation . . . 87
6.2.3 Refcard . . . 87
6.3 5.3 gptcommit . . . 88
6.3.1 Description . . . 88
6.3.2 Installation . . . 88
6.3.3 Refcard . . . 88
6.4 5.4 AI Shell (ai-shell) . . . 89
6.4.1 Description . . . 89
6.4.2 Installation . . . 89
6.4.3 Refcard . . . 89
6.5 5.5 Butterfish . . . 90
6.5.1 Description . . . 90
6.5.2 Installation . . . 90
6.5.3 Refcard . . . 90
6.6 5.6 shell-ai . . . 91
6.6.1 Description . . . 91
6.6.2 Installation . . . 91
6.6.3 Refcard . . . 91
6.7 5.7 Pieces CLI . . . 92
6.7.1 Description . . . 92
6.7.2 Installation . . . 92
6.7.3 Refcard . . . 92
6.8 5.8 promptfoo . . . 93
6.8.1 Description . . . 93
6.8.2 Caractéristiques . . . 93
6.8.3 Installation . . . 93
6.8.4 Refcard . . . 93
6.8.5 Intégration IDE . . . 93
6.8.6 Cas d'usage . . . 93
6.9 5.9 ata (Ask The Terminal Anything) . . . 94
6.9.1 Description . . . 94
6.9.2 Installation . . . 94
6.9.3 Refcard . . . 94
6.10 5.10 hai (Braintrust / variantes) . . . 95
6.10.1 Description . . . 95
6.10.2 Installation . . . 95
6.10.3 Refcard . . . 95
6.11 5.11 Khoj . . . 96
6.11.1 Description . . . 96
6.11.2 Installation . . . 96
6.11.3 Refcard . . . 96
6.11.4 Intégrations . . . 96
6.12 5.12 Tabby . . . 97
6.12.1 Description . . . 97
6.12.2 Installation . . . 97

6.12.3 Refcard . . . 97
6.12.4 Intégration IDE . . . 97
6.13 5.13 Synthèse — Outils spécialisés . . . 98

7 Partie VI — Frameworks d'agents avec usage CLI 99
7.1 6.1 AutoGPT . . . 99
7.1.1 Description . . . 99
7.1.2 Caractéristiques . . . 99
7.1.3 Installation . . . 99
7.1.4 Refcard . . . 100
7.2 6.2 BabyAGI . . . 101
7.2.1 Description . . . 101
7.2.2 Caractéristiques . . . 101
7.2.3 Installation . . . 101
7.3 6.3 CrewAI . . . 102
7.3.1 Description . . . 102
7.3.2 Caractéristiques . . . 102
7.3.3 Installation . . . 102
7.3.4 Refcard . . . 102
7.4 6.4 MetaGPT . . . 103
7.4.1 Description . . . 103
7.4.2 Caractéristiques . . . 103
7.4.3 Installation . . . 103
7.4.4 Refcard . . . 103
7.5 6.5 SuperAGI . . . 104
7.5.1 Description . . . 104
7.5.2 Installation . . . 104
7.5.3 Refcard . . . 104
7.6 6.6 AgentGPT . . . 105
7.6.1 Description . . . 105
7.6.2 Installation . . . 105
7.7 6.7 LangChain CLI . . . 106
7.7.1 Description . . . 106
7.7.2 Installation . . . 106
7.7.3 Refcard . . . 106
7.8 6.8 LangGraph CLI . . . 107
7.8.1 Description . . . 107
7.8.2 Installation . . . 107
7.8.3 Refcard . . . 107
7.8.4 Intégration IDE . . . 107
7.9 6.9 Haystack / Hayhooks . . . 108
7.9.1 Description . . . 108
7.9.2 Caractéristiques . . . 108
7.9.3 Installation . . . 108
7.9.4 Refcard . . . 108
7.10 6.10 Letta (ex-MemGPT) . . . 109
7.10.1 Description . . . 109
7.10.2 Caractéristiques . . . 109
7.10.3 Installation . . . 109
7.10.4 Refcard . . . 109
7.11 6.11 Agno (ex-Phidata) . . . 110
7.11.1 Description . . . 110
7.11.2 Caractéristiques . . . 110
7.11.3 Installation . . . 110
7.11.4 Refcard . . . 110
7.12 6.12 Pydantic AI CLI (clai) . . . 111
7.12.1 Description . . . 111

7.12.2 Caractéristiques . . . 111
7.12.3 Installation . . . 111
7.12.4 Refcard . . . 111
7.13 6.13 Semantic Kernel . . . 112
7.13.1 Description . . . 112
7.13.2 Installation . . . 112
7.13.3 Refcard . . . 112
7.14 6.14 Autres frameworks notables . . . 113

8 Partie VII — Tableau comparatif synthétique 114
8.1 7.1 Vue générale . . . 114
8.2 7.2 Choix d'outil par cas d'usage . . . 117
8.2.1 Refactoring d'un projet existant (15 K lignes) . . . 117
8.2.2 Génération de projet from-scratch . . . 117
8.2.3 Agent autonome long-running . . . 117
8.2.4 Filtre LLM dans un pipeline shell . . . 117
8.2.5 Conversion langage naturel → commande shell . . . 117
8.2.6 Pas de cloud, 100 % local . . . 118
8.2.7 Évaluation et red-team de prompts . . . 118
8.2.8 Génération de messages de commit . . . 118
8.2.9 RAG sur documents personnels . . . 118
8.2.10 Mémoire long-terme persistante . . . 118
8.2.11 Intégration MCP . . . 118
8.3 7.3 Comparatif des modèles économiques . . . 118
8.4 7.4 Sécurité et confidentialité . . . 119

9 Partie VIII — Guide d'intégration IDE 120
9.1 8.1 VSCode et VSCodium . . . 120
9.1.1 Extensions officielles . . . 120
9.1.2 Installation typique . . . 121
9.1.3 Sur VSCodium spécifiquement . . . 121
9.1.4 Terminal intégré . . . 121
9.1.5 MCP (Model Context Protocol) . . . 121
9.2 8.2 JetBrains (IntelliJ, PyCharm, GoLand, WebStorm, Rider, RubyMine, CLion, PhpStorm…) . . . 122
9.2.1 Plugins officiels . . . 122
9.2.2 Installation via CLI . . . 122
9.2.3 Plus la voie du terminal . . . 122
9.2.4 MCP Server JetBrains . . . 122
9.3 8.3 Neovim / Vim . . . 123
9.3.1 Plugins recommandés . . . 123
9.3.2 Exemple : configuration minimale avec lazy.nvim . . . 123
9.3.3 Astuce : tmux + agent CLI . . . 123
9.4 8.4 Emacs . . . 124
9.4.1 Exemple : configuration `gptel` . . . 124
9.4.2 Emacs + LLM via tramp . . . 124
9.5 8.5 Helix . . . 125
9.6 8.6 Sublime Text et autres . . . 126
9.7 8.7 Recettes types . . . 127
9.7.1 Workflow A — Aider dans tmux à côté de Neovim . . . 127
9.7.2 Workflow B — VSCode + Continue + Ollama local . . . 127
9.7.3 Workflow C — JetBrains + MCP + Claude Code en terminal externe . . . 127
9.7.4 Workflow D — DevOps : générer un message de commit puis pousser . . . 127

10 Partie IX — Lexique technique 128
10.1 A . . . 128
10.2 B . . . 128
10.3 C . . . 128

10.4 D . . . 129
10.5 E . . . 129
10.6 F . . . 129
10.7 G . . . 129
10.8 H . . . 129
10.9 I . . . 129
10.10J . . . 129
10.11L . . . 129
10.12M . . . 130
10.13O . . . 130
10.14P . . . 130
10.15Q . . . 130
10.16R . . . 130
10.17S . . . 130
10.18T . . . 131
10.19V . . . 131
10.20W . . . 131
10.21Y . . . 131
10.22Z . . . 131

11 Partie X — Liens et ressources complémentaires 132
11.1 10.1 Sites officiels et dépôts GitHub . . . 132
11.1.1 CLI propriétaires majeurs . . . 132
11.1.2 Agents codeurs OSS . . . 132
11.1.3 CLI LLM génériques . . . 133
11.1.4 Runners locaux . . . 133
11.1.5 Outils spécialisés . . . 133
11.1.6 Frameworks d'agents . . . 134
11.2 10.2 Modèles et fournisseurs LLM . . . 134
11.3 10.3 Spécifications et standards . . . 134
11.4 10.4 Benchmarks publics . . . 134
11.5 10.5 Listes / catalogues communautaires . . . 135
11.6 10.6 Communauté francophone . . . 135
11.7 10.7 Documentation et apprentissage . . . 135
11.8 10.8 Newsletters et fils d'actualité . . . 135
11.9 10.9 Outils complémentaires . . . 135
11.1010.10 Sites officiels Anthropic / OpenAI / Google détaillés . . . 136

12 Conclusion 137

13 Annexe — Note méthodologique 138

Chapter 1

Avant-propos

Ce guide propose un panorama aussi exhaustif que possible des outils d'intelligence artificielle utilisables en ligne de commande (CLI, *Command Line Interface*). Il est destiné aux administrateurs systèmes, développeurs, ingénieurs DevOps, chercheurs et utilisateurs avancés de Linux qui souhaitent intégrer l'IA générative directement dans leurs flux de travail terminal.

L'écosystème des CLI IA explose littéralement depuis 2023 : agents de codage autonomes, runners de modèles locaux, wrappers shell pour LLM, frameworks d'agents multi-tâches… Chaque mois apporte son lot de nouveaux outils et chaque outil évolue rapidement. Ce document tente de couvrir l'état de l'art à mi-2026 en regroupant plus de **quarante outils majeurs** dans une structure cohérente.

1.1 Comment lire ce guide

Chaque outil est présenté sous une fiche technique standardisée comportant :

- **Identité** : nom officiel, nom du binaire, éditeur, pays d'origine, première sortie, licence ;
- **Description** : nature et positionnement de l'outil ;
- **Caractéristiques techniques** : modèles supportés, mode agentique, langages cibles, capacités spécifiques ;
- **Installation** : commandes pour Debian/Ubuntu et Fedora/RHEL ;
- **Refcard** : liste des sous-commandes, options et flags principaux ;
- **Intégrations IDE** : VSCode/VSCodium, JetBrains, Neovim, Emacs ;
- **Liens** : site officiel, dépôt GitHub, documentation.

1.2 Mise en garde sur les versions

Les numéros de versions exacts indiqués correspondent à des estimations basées sur la trajectoire connue jusqu'au début 2026. **Vérifiez systématiquement la dernière version stable** sur le site officiel ou le dépôt GitHub avant installation. Les commandes d'installation, en revanche, sont généralement stables sur plusieurs cycles de release.

1.3 Conventions typographiques

- `commande` : nom de binaire ou commande shell exacte ;
- *italique* : terme étranger ou concept à définir ;
- **gras** : élément clé ou nom propre d'un outil ;
- Les blocs de code sont préfixés par le shell concerné lorsque c'est utile.

1.4 Catégories couvertes

Le guide est organisé en six familles :

1. **CLI IA propriétaires majeurs** — Claude Code, OpenAI Codex CLI, Google Gemini CLI, GitHub Copilot CLI, Cursor, Cody, Amazon Q, Warp AI…
2. **Agents de codage open-source** — Aider, Plandex, Goose, OpenHands, Continue, Cline, Open Interpreter, gpt-engineer, gptme, SWE-agent, Mentat, Smol Developer…
3. **CLI LLM génériques** — llm, ShellGPT, aichat, mods, tgpt, fabric, gptscript, smartcat, gorilla-cli…
4. **Runners de modèles locaux** — Ollama, llama.cpp, LM Studio CLI, llamafile, vLLM, GPT4All, LocalAI, RamaLama, whisper.cpp, piper…
5. **Outils spécialisés** — aicommits, opencommit, gptcommit, AI Shell, Butterfish, promptfoo, Pieces CLI…
6. **Frameworks d'agents** — AutoGPT, BabyAGI, CrewAI, MetaGPT, SuperAGI, LangChain CLI, LangGraph CLI, Letta/MemGPT…

À la fin du document figurent un **tableau comparatif synthétique**, un **guide d'intégration IDE détaillé**, un **lexique technique** et une **liste de liens** pour aller plus loin.

Chapter 2

Partie I — CLI IA propriétaires majeurs

Cette première partie couvre les outils CLI développés par les grands éditeurs commerciaux d'IA générative. Ce sont aujourd'hui les outils les plus matures, les plus performants en agentique, mais aussi ceux dont l'utilisation implique souvent un abonnement payant et l'envoi de code sur des serveurs distants.

2.1 1.1 Claude Code (Anthropic)

Nom du binaire : `claude` **Éditeur** : Anthropic (États-Unis, San Francisco) **Pays d'origine** : USA **Première sortie publique** : février 2025 (research preview), GA en mai 2025 **Licence** : propriétaire (CLI client open-source dans certaines parties, modèles propriétaires) **Site officiel** : https://www.claude.com/product/claude-code **Dépôt GitHub** : https://github.com/anthropics/claude-code

2.1.1 Description

Claude Code est l'agent CLI officiel d'Anthropic propulsé par les modèles **Claude Sonnet 4.x** et **Claude Opus 4.x**. C'est un agent de codage agentique capable d'éditer du code, d'exécuter des commandes shell, de lancer des tests, de naviguer dans une base de code et de coordonner des sous-tâches. Il se distingue par sa très grande fenêtre de contexte (jusqu'à 1 million de tokens sur Opus 4.7) et son intégration native du protocole **MCP** (*Model Context Protocol*) qu'Anthropic a inventé.

2.1.2 Caractéristiques

- Modèles : Claude Opus 4.6/4.7, Sonnet 4.5/4.6, Haiku 4.5
- Contexte : jusqu'à 1 M tokens (Opus 4.7 mode 1M)
- Agentique multi-étapes avec planification et sous-agents
- Système d'**hooks** (pre-tool, post-tool, stop, user-prompt-submit) configurables dans `~/.claude/settings.json`
- **Slash commands** personnalisables, **skills** et **agents** customs
- Support natif **MCP** (clients et serveurs)
- Intégration **IDE** via extensions VSCode/JetBrains officielles
- Mode plan (`/plan`), mode auto-accept (`shift+tab`), permission-mode configurables
- Sandbox optionnelle via Docker / `claude-code-sandbox`

2.1.3 Installation

Debian/Ubuntu :

```
# Pré-requis : Node.js >= 18
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs
sudo npm install -g @anthropic-ai/claude-code
```

```
# Authentification
claude  # ouvre le navigateur pour OAuth Anthropic Console
```

Fedora/RHEL :

```
sudo dnf install -y nodejs npm
sudo npm install -g @anthropic-ai/claude-code
claude
```

Méthode alternative (universelle, sans Node) :

```
curl -fsSL https://claude.ai/install.sh | bash
```

2.1.4 Refcard des commandes

Commande / Slash	Description
`claude`	démarre une session interactive dans le répertoire courant
`claude -p "prompt"`	mode print non-interactif (one-shot)
`claude --resume`	reprendre la dernière session
`claude --continue`	continuer la dernière session (alias `-c`)
`claude config`	gestion de la configuration
`claude mcp`	gestion des serveurs MCP
`/help`	aide intégrée
`/clear`	nettoie le contexte
`/compact`	compacte la conversation
`/cost`	affiche le coût en tokens
`/permissions`	gère les permissions d'outils
`/agents`	liste / crée des sous-agents
`/mcp`	liste les serveurs MCP actifs
`/init`	génère un fichier `CLAUDE.md` du projet
`/review`	review de code
`/plan`	bascule en mode plan
`Shift+Tab`	bascule en mode auto-accept
`Esc`	interrompt l'agent
`!commande`	exécute commande shell dans le contexte

2.1.5 Intégration IDE

- **VSCode / VSCodium** : extension officielle « Claude Code for VS Code », active la communication entre l'éditeur ouvert et la CLI (sélections, diagnostics, ouverture de fichiers).
- **JetBrains** (IntelliJ, PyCharm, GoLand, WebStorm, Rider…) : plugin officiel disponible sur le JetBrains Marketplace.
- **Neovim / Vim** : intégrable via `claude-code.nvim` (communauté).
- **Emacs** : `claude-code.el` (communauté).
- **Terminal** : autonome, fonctionne dans n'importe quel terminal moderne.

2.1.6 Modèle économique

Inclus dans les abonnements Claude Pro (~20 $/mois), Max (~100 $/mois), ainsi que par les plans API à l'usage. Quotas fonction du plan. Disponible aussi via **Amazon Bedrock** et **Google Vertex AI**.

2.2 1.2 OpenAI Codex CLI

Nom du binaire : `codex` **Éditeur** : OpenAI (USA, San Francisco) **Pays d'origine** : USA **Première sortie** : avril 2025 (réécrit en Rust en juin 2025) **Licence** : Apache 2.0 (client) — modèles propriétaires **Site officiel** : https://developers.openai.com/codex/cli **Dépôt GitHub** : https://github.com/openai/codex

2.2.1 Description

Codex CLI est l'agent terminal d'OpenAI, exécutant les modèles **GPT-5 Codex**, **o4-mini**, **o3** et **GPT-4.1** localement avec un sandboxing renforcé. Initialement écrit en TypeScript/Node, il a été entièrement réécrit en **Rust** en 2025 pour de meilleures performances et une isolation plus solide (Seatbelt sur macOS, Landlock sur Linux). Il vise la parité fonctionnelle avec Claude Code en s'intégrant à l'écosystème OpenAI.

2.2.2 Caractéristiques

- Modèles : GPT-5 / GPT-5 Codex, o-series (raisonnement), GPT-4.1
- Sandbox Linux via **Landlock + seccomp** (mode `--full-auto`)
- Multimodal : peut lire des images / captures d'écran fournies en input
- Modes d'approbation : `--ask-for-approval`, `--auto-edit`, `--full-auto`
- Configuration via `~/.codex/config.toml`
- Profils multi-modèles
- Support MCP (depuis fin 2025)
- Headless mode pour CI/CD : `codex exec`

2.2.3 Installation

Debian/Ubuntu :

```
# Via npm (toujours disponible comme wrapper)
sudo npm install -g @openai/codex

# OU via Homebrew on Linux
brew install codex

# OU binaire Rust direct
curl -fsSL https://github.com/openai/codex/releases/latest/download/codex-x86_64-unknown-linux-gnu.tar.gz
  | tar -xz -C /tmp && sudo mv /tmp/codex /usr/local/bin/
```

Fedora :

```
sudo dnf install -y nodejs npm
sudo npm install -g @openai/codex
# Authentification
codex login
```

2.2.4 Refcard

Commande	Description
`codex`	session interactive (TUI)
`codex "prompt"`	session avec prompt initial
`codex exec "prompt"`	mode non-interactif (CI)
`codex --model gpt-5-codex`	choix du modèle
`codex --ask-for-approval`	demande confirmation avant chaque action
`codex --auto-edit`	édition auto, commandes shell soumises à approbation
`codex --full-auto`	mode entièrement autonome (sandboxed)
`codex --image screenshot.png "..."`	input image
`codex login` / `logout`	authentification ChatGPT/API

Commande	Description
`codex resume`	reprendre une session
`codex --config provider=openai`	override config

2.2.5 Intégration IDE

- **VSCode** : extension « OpenAI Codex » officielle (mai 2025), avec mode chat latéral et exécution dans terminal.
- **JetBrains** : plugin OpenAI officiel disponible.
- **Cursor** et **Windsurf** : intègrent Codex en backend de leurs propres agents.
- **Neovim** : `codex.nvim` (officiel).

2.2.6 Tarification

Inclus dans **ChatGPT Plus** (20 $), **Pro** (200 $) et **Business**. Aussi disponible via la **Platform OpenAI** à l'usage (tokens). Quota mensuel limité sur Plus, illimité sur Pro.

2.3 1.3 Google Gemini CLI

Nom du binaire : `gemini` **Éditeur** : Google / Google DeepMind (USA / UK) **Pays d'origine** : USA **Première sortie** : juin 2025 **Licence** : Apache 2.0 **Site officiel** : https://cloud.google.com/gemini/docs/codeassist/gemini-cli **Dépôt GitHub** : https://github.com/google-gemini/gemini-cli

2.3.1 Description

Gemini CLI est l'agent terminal officiel de Google, propulsé par les modèles **Gemini 2.5 Pro** et **Gemini 3 Pro**. Il offre une fenêtre de contexte allant jusqu'à 2 millions de tokens, exploite la recherche Google intégrée, et propose un palier gratuit généreux (1000 requêtes/jour avec compte Google personnel). Il s'intègre nativement avec **Vertex AI**, **Gemini Code Assist** et l'écosystème Google Cloud.

2.3.2 Caractéristiques

- Modèles : Gemini 2.5 Pro / Flash, Gemini 3 Pro
- Contexte : jusqu'à 2 M tokens
- Recherche Google intégrée (grounding)
- Multimodal : images, PDF, audio en input
- Support MCP (depuis fin 2025)
- Système de **commandes personnalisables** (`.gemini/commands/`)
- Mode YOLO (`--yolo`) entièrement autonome
- Sandbox via Docker/Podman (`--sandbox`)

2.3.3 Installation

Debian/Ubuntu :

```
# Pré-requis : Node.js 20+
sudo npm install -g @google/gemini-cli

# Authentification (3 méthodes)
gemini  # OAuth Google personnel
# OU
export GEMINI_API_KEY="votre_clé"
# OU
gcloud auth application-default login  # Vertex AI
```

Fedora :

```
sudo dnf install -y nodejs npm
sudo npm install -g @google/gemini-cli
```

Homebrew :

```
brew install gemini-cli
```

2.3.4 Refcard

Commande / Slash	Description
`gemini`	session interactive
`gemini -p "prompt"`	non-interactif
`gemini --model gemini-2.5-pro`	choix du modèle
`gemini --sandbox`	exécution sandboxée
`gemini --yolo`	mode autonome sans confirmation
`gemini --checkpointing`	snapshots de session
`/help`, `/quit`	navigation
`/clear`, `/compress`	gestion contexte

Commande / Slash	Description
`/tools`	liste les outils MCP
`/memory`	gestion mémoire long-terme
`/chat save <name>`	sauvegarde de session
`/chat resume <name>`	reprise
`@chemin/fichier`	inclut un fichier dans le prompt

2.3.5 Intégration IDE

- **VSCode / VSCodium** : extension « Gemini Code Assist » officielle, avec chat, autocomplete et lien vers la CLI.
- **JetBrains** : plugin Gemini Code Assist officiel.
- **IntelliJ Cloud Tools** : pré-intégré dans Cloud Code.
- **Neovim** : intégration via `gemini.nvim`.

2.3.6 Tarification

- **Gratuit** (compte Google) : 1000 req/jour, 60 req/min.
- **Code Assist Standard** : ~19 $/mois.
- **Code Assist Enterprise** : ~45 $/mois.
- **Vertex AI** : à l'usage (tokens).

2.4 1.4 GitHub Copilot CLI

Nom du binaire : `gh copilot` (extension de `gh`) et `copilot` (CLI standalone depuis 2025) **Éditeur** : GitHub (filiale Microsoft) **Pays d'origine** : USA **Première sortie** : mars 2023 (preview), GA en novembre 2023, version standalone agentique en septembre 2025 **Licence** : propriétaire (extension `gh-copilot` est public source) **Site officiel** : https://github.com/features/copilot **Dépôt** : https://github.com/github/gh-copilot

2.4.1 Description

GitHub Copilot CLI a deux variantes :

1. **`gh copilot`** : extension de la CLI GitHub historique, propose `suggest` (générer une commande) et `explain` (expliquer une commande shell).
2. **`copilot`** (depuis sept. 2025) : agent terminal complet façon Claude Code, capable d'éditer du code, exécuter des tests, gérer les PR, propulsé par GPT-5, Claude Sonnet 4.5 ou Gemini 2.5.

2.4.2 Caractéristiques

- Multi-modèles : GPT-5, GPT-4.1, Claude Sonnet 4.5, Gemini 2.5 Pro (sélection au runtime)
- Intégration native **GitHub** : issues, PR, gh API, Actions
- Support **MCP**
- Mode `--allow-all-tools`, sandbox par défaut
- Authentification via `gh auth login`
- Programmable headless via `copilot exec`

2.4.3 Installation

Debian/Ubuntu :

```
# 1) Installer gh
sudo apt update && sudo apt install -y gh
gh auth login

# 2a) Extension legacy
gh extension install github/gh-copilot

# 2b) Nouveau CLI agentique standalone
sudo npm install -g @github/copilot
copilot
```

Fedora :

```
sudo dnf install -y gh nodejs npm
gh auth login
gh extension install github/gh-copilot
sudo npm install -g @github/copilot
```

2.4.4 Refcard

Commande	Description
`gh copilot suggest "tâche"`	suggère une commande shell
`gh copilot explain "cmd"`	explique une commande
`copilot`	session agentique interactive
`copilot -p "prompt"`	one-shot
`copilot --model claude-sonnet-4.5`	choix modèle
`copilot --allow-all-tools`	autorise outils sans demander
`copilot mcp`	gestion MCP

Commande	Description
`/agents`, `/list-tools`	introspection

2.4.5 Intégration IDE

- **VSCode** : Copilot natif (chat + complétion + agents).
- **JetBrains** : plugin GitHub Copilot officiel.
- **Neovim** : `copilot.vim` officiel + `CopilotChat.nvim`.
- **Visual Studio**, **Eclipse**, **Xcode** : plugins officiels.

2.4.6 Tarification

- **Free** : 50 messages chat/mois + 2000 complétions.
- **Pro** : 10 $/mois.
- **Business** : 19 $/utilisateur/mois.
- **Enterprise** : 39 $/utilisateur/mois.

2.5 1.5 Cursor CLI / Cursor Agent

Nom du binaire : `cursor-agent` **Éditeur** : Anysphere (USA) **Pays d'origine** : USA **Première sortie CLI** : août 2025 **Licence** : propriétaire **Site officiel** : https://cursor.com/cli **Documentation** : https://docs.cursor.com/cli

2.5.1 Description

Cursor a longtemps été un fork de VSCode boosté à l'IA. Depuis août 2025, Anysphere propose **Cursor Agent CLI**, une version terminal de son agent qui peut tourner en headless dans une CI, dans un container distant, ou comme outil terminal indépendant. Il partage le moteur d'agent et les modèles entraînés/sélectionnés avec l'IDE Cursor.

2.5.2 Caractéristiques

- Multi-modèles : GPT-5, Claude Sonnet 4.5, Gemini 2.5, modèle « Cursor » maison
- Mode background : peut tourner en arrière-plan dans le cloud Cursor (tâches longues)
- Hooks et règles via `.cursor/rules/`
- Support MCP
- Synchronisation de session avec l'IDE Cursor

2.5.3 Installation

Debian/Ubuntu / Fedora :

```
curl https://cursor.com/install -fsS | bash
# Ajoute ~/.local/bin au PATH si besoin
cursor-agent login
```

2.5.4 Refcard

Commande	Description
`cursor-agent`	session TUI
`cursor-agent -p "prompt"`	one-shot
`cursor-agent --print`	mode print pour scripts
`cursor-agent --model claude-sonnet-4.5`	choix modèle
`cursor-agent login` / `logout`	authentification
`cursor-agent --background`	tâche en arrière-plan cloud
`cursor-agent resume <id>`	reprise de session

2.5.5 Intégration IDE

- **Cursor** (l'IDE) : natif, partage la session avec la CLI.
- **VSCode / VSCodium** : extension « Cursor Connector » non-officielle.
- **JetBrains** : non supporté nativement (utiliser MCP).

2.5.6 Tarification

- **Free** : limites strictes.
- **Pro** : 20 $/mois.
- **Business** : 40 $/utilisateur/mois.
- **Ultra** : 200 $/mois (nouveau plan 2025).

2.6 1.6 Sourcegraph Cody CLI

Nom du binaire : `cody` **Éditeur** : Sourcegraph (USA) **Pays d'origine** : USA **Première sortie** : 2024 **Licence** : Apache 2.0 (CLI) **Site officiel** : https://sourcegraph.com/cody **Dépôt** : https://github.com/sourcegraph/cody

2.6.1 Description

Cody est l'assistant IA de Sourcegraph, particulièrement efficace pour les **grandes bases de code** grâce à l'indexation et la recherche sémantique de Sourcegraph. La CLI permet de poser des questions en langage naturel sur un repo, générer du code, expliquer du code et naviguer dans des contextes très larges (millions de fichiers).

2.6.2 Caractéristiques

- Multi-modèles : Claude, GPT-4, Gemini, Mixtral, modèles locaux (via API)
- Indexation sémantique de code (graphCode)
- Mode entreprise : connecte plusieurs repos, gestion centralisée
- Support de prompts personnalisés (`.cody/prompts/`)

2.6.3 Installation

Debian/Ubuntu / Fedora :

```
sudo npm install -g @sourcegraph/cody
cody auth login
```

Binaire :

```
curl -fsSL https://sourcegraph.com/.api/cody/cli/install.sh | sh
```

2.6.4 Refcard

Commande	Description
`cody chat -m "prompt"`	message simple
`cody chat --context-file FILE`	inclut fichier dans contexte
`cody chat --context-repo OWNER/REPO`	contexte d'un repo Sourcegraph
`cody auth login`	authentification
`cody api`	accès direct à l'API

2.6.5 Intégration IDE

- **VSCode** : extension officielle Cody.
- **JetBrains** : plugin officiel Cody.
- **Neovim** : `sg.nvim` (Sourcegraph officiel).
- **Visual Studio**, **Eclipse** : non officiel.

2.6.6 Tarification

- **Free** : 200 chats/mois, 500 autocomplétions.
- **Pro** : 9 $/mois.
- **Enterprise** : sur devis (intégration LDAP/SSO, on-premise).

2.7 1.7 Amazon Q Developer CLI

Nom du binaire : `q` **Éditeur** : Amazon Web Services **Pays d'origine** : USA **Première sortie** : avril 2024 (renommée depuis CodeWhisperer CLI) **Licence** : propriétaire **Site officiel** : https://aws.amazon.com/q/developer/ **Documentation CLI** : https://docs.aws.amazon.com/amazonq/latest/qdeveloper-ug/command-line.html

2.7.1 Description

Amazon Q Developer (anciennement **CodeWhisperer**) est l'assistant IA d'AWS pour les développeurs et les ingénieurs cloud. La CLI propose à la fois un agent chat, des autocomplétions de commandes shell, et une intégration profonde avec **AWS** (debug d'erreurs CloudFormation, génération de policies IAM, exploration de comptes AWS via langage naturel).

2.7.2 Caractéristiques

- Modèle : Claude (via Bedrock)
- Mode chat agentique avec accès filesystem et shell
- Auto-complétion shell (zsh, bash, fish) intelligente AWS-aware
- Intégration MCP
- Translation langage naturel → commandes AWS CLI
- Sécurité : peut s'auto-contraindre par profil IAM

2.7.3 Installation

Debian/Ubuntu :

```
curl --proto '=https' --tlsv1.2 -sSf \
  https://desktop-release.q.us-east-1.amazonaws.com/latest/amazon-q.deb -o amazon-q.deb
sudo apt install -y ./amazon-q.deb
q login
```

Fedora :

```
sudo rpm -i https://desktop-release.q.us-east-1.amazonaws.com/latest/amazon-q.rpm
q login
```

Headless / sans GUI :

```
curl --proto '=https' --tlsv1.2 -sSf \
  https://desktop-release.q.us-east-1.amazonaws.com/latest/q-x86_64-linux.zip -o q.zip
unzip q.zip && cd q && ./install.sh --no-confirm
```

2.7.4 Refcard

Commande	Description
`q chat`	session chat
`q translate "..."`	NL → commande shell
`q doctor`	diagnostic
`q login` / `logout`	authentification AWS Builder ID / IAM
`q settings`	config
`q whoami`	identité courante
`q theme`	thème terminal
`q ai "explain this error: ..."`	aide ad-hoc

2.7.5 Intégration IDE

- **VSCode** : extension Amazon Q officielle.

- **JetBrains** : plugin Amazon Q officiel.
- **Visual Studio**, **Eclipse** : extensions officielles.

2.7.6 Tarification

- **Free Tier** : 50 chats/mois, autocomplétion shell illimitée.
- **Pro** : 19 $/utilisateur/mois (intégrations AWS avancées, codebases privées).

2.8 1.8 Warp Terminal AI

Nom du binaire : intégré à `warp` (terminal complet) **Éditeur** : Warp (USA) **Pays d'origine** : USA **Première sortie de l'IA** : 2023, agent autonome en 2025 **Licence** : propriétaire (terminal gratuit, fonctions IA freemium) **Site officiel** : https://www.warp.dev

2.8.1 Description

Warp est un **terminal moderne** entièrement repensé (UI bloc-par-bloc, palette de commandes, partage d'output) qui intègre nativement un agent IA. Depuis 2025, l'agent Warp peut prendre des décisions, lancer des commandes en chaîne et travailler comme un agent CLI complet — sans qu'il y ait besoin d'un binaire séparé : c'est le terminal lui-même qui devient l'interface IA.

2.8.2 Caractéristiques

- Modèles : Claude, GPT, Gemini sélectionnables
- Auto-complétion et correction de commandes
- Agent autonome avec mode YOLO
- Workflows partagés en équipe (Warp Drive)
- Disponible Linux/macOS/Windows

2.8.3 Installation

Debian/Ubuntu :

```
wget https://app.warp.dev/download?package=deb -O warp.deb
sudo apt install -y ./warp.deb
```

Fedora :

```
sudo rpm -i https://app.warp.dev/download?package=rpm
```

2.8.4 Refcard

L'IA s'invoque par `#` au début d'une commande (ex: `# trouve-moi tous les .log de plus de 1Go`) ou via le bouton **Warp AI** dans l'interface. Les workflows sont définis en YAML.

2.8.5 Intégration IDE

Warp est lui-même un terminal autonome. Il s'utilise en complément des IDE plutôt qu'en intégration.

2.8.6 Tarification

- **Free** : 150 requêtes IA/mois.
- **Pro** : 15 $/utilisateur/mois.
- **Team / Enterprise** : 22 $ et plus.

2.9 1.9 Tabnine CLI

Nom du binaire : `tabnine` (CLI complémentaire des plugins IDE) **Éditeur** : Tabnine (Israël) **Pays d'origine** : Israël **Première sortie** : 2018 (plugin), CLI agent en 2024 **Licence** : propriétaire **Site officiel** : https://www.tabnine.com

2.9.1 Description

Tabnine est l'un des pionniers du *AI code completion*, fondé en Israël avant l'arrivée de Copilot. Aujourd'hui Tabnine propose un **agent chat CLI** principalement destiné aux entreprises : le focus est sur les déploiements **on-premise**, la confidentialité, et l'entraînement personnalisé sur les bases de code privées des clients.

2.9.2 Caractéristiques

- Modèles : « Tabnine Protected » (entraînement uniquement sur licences permissives), Claude/GPT en option
- Déploiement on-premise / air-gapped possible
- Personnalisation par fine-tuning client

2.9.3 Installation

```
# Plugin VSCode/JetBrains via marketplace ; CLI standalone
sudo npm install -g @tabnine/cli
tabnine login
```

2.9.4 Intégration IDE

- **VSCode**, **JetBrains** (toute la suite), **Vim/Neovim**, **Emacs**, **Sublime**, **Visual Studio**, **Eclipse** : plugins officiels.

2.9.5 Tarification

- **Dev** : 9 $/mois.
- **Enterprise** : 39 $/utilisateur/mois (on-prem, fine-tune).

2.10 1.10 Autres CLI propriétaires notables

- **Replit Agent** : agent dans le navigateur Replit, propose une CLI `replit` mais surtout un IDE web. https://replit.com
- **Windsurf CLI** (Codeium) : devenu *Cascade*, agent terminal lié à l'IDE Windsurf. https://codeium.com/windsurf
- **JetBrains AI Assistant** : pas de CLI dédiée à proprement parler, mais accessible via le terminal intégré IDE et via plugin AI Assistant. https://www.jetbrains.com/ai/
- **Hugging Chat CLI** (`huggingface_hub` chat-cli) : non-commercial, sera traité dans la partie open-source.
- **Perplexity CLI** : `pplx` ou `perplexity-cli` (non-officiel mais largement utilisé).
- **Mistral Le Chat CLI** : `mistral-cli` ou clients tiers (Mistral est français et propose une API).

Chapter 3

Partie II — Agents de codage open-source

Cette partie couvre les agents CLI ouverts (généralement MIT, Apache ou AGPL) qui se positionnent comme alternatives ou compléments aux agents propriétaires. Ils partagent en général la propriété de pouvoir utiliser plusieurs *backends* LLM (cloud ou locaux), ce qui les rend plus flexibles et moins enfermants.

3.1 2.1 Aider

Nom du binaire : `aider` **Auteur** : Paul Gauthier (USA) **Pays d'origine** : USA **Première sortie** : mai 2023 **Licence** : Apache 2.0 **Site officiel** : https://aider.chat **Dépôt GitHub** : https://github.com/Aider-AI/aider (anciennement `paul-gauthier/aider`)

3.1.1 Description

Aider est sans doute l'agent CLI open-source le plus populaire et mature. Écrit en Python, il s'intègre directement à `git` (chaque modification donne lieu à un commit automatique), supporte une vingtaine de modèles LLM (OpenAI, Anthropic, Gemini, DeepSeek, modèles locaux via Ollama/LiteLLM), et propose plusieurs *modes* (chat, edit, architect, ask). Il est régulièrement classé en tête des benchmarks SWE-bench parmi les outils OSS.

3.1.2 Caractéristiques

- Écrit en Python (3.10+), distribué via PyPI
- Multi-modèles via **LiteLLM** (200+ providers)
- Modes : `/chat`, `/ask`, `/architect`, `/code`
- **Auto-commits Git** avec messages générés par le LLM
- Mode **architect** : un modèle « réflexif » (Claude/o3) planifie, un modèle « éditeur » (Sonnet/Haiku) applique
- Support « repo map » (graphe d'arborescence sémantique du projet)
- Voice mode (`/voice`) avec Whisper
- Édition en *unified diff* ou *whole file* selon le modèle
- Affichage *diff colorisé* dans le terminal

3.1.3 Installation

Debian/Ubuntu :

```
sudo apt install -y python3-pip python3-venv pipx
pipx ensurepath
pipx install aider-chat
# Ou méthode officielle (recommandée) :
curl -LsSf https://aider.chat/install.sh | sh
```

Fedora :

```
sudo dnf install -y pipx
pipx ensurepath
pipx install aider-chat
```

Configuration :

```
export ANTHROPIC_API_KEY="..."
# ou OPENAI_API_KEY, GEMINI_API_KEY, DEEPSEEK_API_KEY...
cd mon-projet
aider
```

3.1.4 Refcard

Commande	Description
`aider`	démarre une session dans le repo courant
`aider fichier1.py fichier2.py`	démarre avec ces fichiers en contexte
`aider --model claude-sonnet-4.5`	choix du modèle
`aider --architect --editor-model gpt-4.1`	mode architecte
`aider --weak-model gemini-flash`	modèle faible pour les commits
`aider --no-auto-commits`	désactive auto-commit
`aider --watch-files`	recharge à chaque modif filesystem
`aider --browser`	UI web locale
`/add fichier`	ajoute un fichier au contexte
`/drop fichier`	retire du contexte
`/diff`	affiche le diff
`/commit`	commit manuel
`/undo`	annule le dernier commit
`/run cmd`	exécute une commande shell
`/test`	exécute la commande de test
`/ask`	question sans modification
`/code`	passe en mode édition
`/architect`	passe en mode architecte
`/voice`	enregistrement vocal
`/help`	aide

3.1.5 Intégration IDE

- **VSCode / VSCodium** : extension communautaire « Aider VSCode » + ouverture via terminal intégré.
- **JetBrains** : plugin communautaire « Aider » et utilisation via terminal.
- **Neovim** : `aider.nvim`, `nvim-aider` (deux plugins concurrents).
- **Emacs** : `aider.el` / `aidermacs`.
- **Vim** : intégration via `vim-aider`.
- **Tmux** : très souvent utilisé en panneau dédié.

3.1.6 Tarification

- Le client est **gratuit et open-source**.
- Coût = appels API au modèle choisi (DeepSeek très bon marché ~ centimes/conversation, Claude Opus plus cher).
- Aucune limitation logicielle, aucun lock-in.

3.2 2.2 Plandex

Nom du binaire : `plandex` (alias `pdx`) **Auteur** : Dane Schneider (USA) **Pays d'origine** : USA **Première sortie** : mars 2024 (v1), v2 réécriture début 2025 **Licence** : MIT (client) / AGPL v3 (serveur self-hosted) **Site officiel** : https://plandex.ai **Dépôt GitHub** : https://github.com/plandex-ai/plandex

3.2.1 Description

Plandex est un agent terminal orienté **tâches longues et grands projets**. Sa particularité est son architecture client/serveur : un serveur Go (auto-hébergeable) gère les *plans* (sessions), les *branches* (variantes parallèles) et la persistance ; le client CLI n'est qu'une interface. Il introduit un **mode auto-debug** très abouti (boucles tests/debug autonomes).

3.2.2 Caractéristiques

- Architecture client/serveur (PostgreSQL + Go)
- Plans persistants, branches, contextes versionnés
- Auto-debug (réexécute les tests jusqu'à passage)
- Sandbox via Docker
- Multi-modèles (OpenAI, Anthropic, Gemini, DeepSeek, OpenRouter, Ollama)
- *Smart context management* : ne charge que ce qui est pertinent
- Mode **full-auto** ou **chat** ou **safe**
- API REST côté serveur

3.2.3 Installation

Debian/Ubuntu / Fedora (client + serveur cloud) :

```
curl -sL https://plandex.ai/install.sh | bash
```

Self-hosted serveur :

```
git clone https://github.com/plandex-ai/plandex
cd plandex/app
docker compose up -d
# Puis pointer le client :
export PLANDEX_API_HOST=http://localhost:8099
plandex sign-in
```

3.2.4 Refcard

Commande	Description
`plandex new`	crée un nouveau plan
`plandex tell "tâche"`	donne une instruction au plan courant
`plandex chat "..."`	discussion sans modif
`plandex load fichier`	ajoute fichier au contexte
`plandex ls`	liste le contexte
`plandex apply`	applique les modifs en pending
`plandex reject`	rejette les modifs
`plandex diff`	diff des modifs
`plandex branches`	liste les branches
`plandex checkout branch`	change de branche
`plandex log`	journal du plan
`plandex models`	gestion modèles
`plandex set-model planner claude-sonnet-4.5`	configure modèle
`plandex debug`	mode auto-debug
`plandex --auto full`	bascule mode full-auto

3.2.5 Intégration IDE

Plandex est principalement *terminal-centric*, mais on peut le combiner avec n'importe quel IDE car il modifie le filesystem (les modifs apparaissent dans VSCode/JetBrains).

3.2.6 Tarification

- **Client/serveur** : open-source gratuit.
- **Plandex Cloud** (hébergé) : palier gratuit + plans payants pour usage intensif.

3.3 2.3 Goose (Block / Square)

Nom du binaire : `goose` **Auteur** : Block, Inc. (Square, ex-Twitter Inc.) **Pays d'origine** : USA **Première sortie** : septembre 2024 **Licence** : Apache 2.0 **Site officiel** : https://block.github.io/goose/ **Dépôt GitHub** : https://github.com/block/goose

3.3.1 Description

Goose est l'agent open-source publié par Block (la société de Jack Dorsey). Écrit en Rust pour la performance, il met l'accent sur l'**extensibilité par MCP** : chaque outil est un serveur MCP (filesystem, computer-controller, GitHub, JetBrains, etc.). Il est multi-modèles, multi-plateformes (Linux, macOS, Windows), et propose un mode CLI ainsi qu'une UI desktop optionnelle.

3.3.2 Caractéristiques

- Écrit en Rust (binaire unique)
- **MCP first** : tous les outils sont des serveurs MCP
- Multi-modèles : Anthropic, OpenAI, Google, Groq, Ollama, OpenRouter, Bedrock, Databricks…
- Sessions nommées et reprenables
- Mode plan, mode auto
- *Recipes* (workflows scriptés)
- UI desktop GTK (optionnelle)

3.3.3 Installation

Debian/Ubuntu / Fedora :

```
curl -fsSL https://github.com/block/goose/releases/download/stable/download_cli.sh | bash
# ou installation manuelle :
wget https://github.com/block/goose/releases/latest/download/goose-x86_64-unknown-linux-gnu.tar.bz2
tar -xjf goose-*.tar.bz2 && sudo mv goose /usr/local/bin/
```

Configuration :

```
goose configure   # configure provider + clé API
goose session     # démarre une session
```

3.3.4 Refcard

Commande	Description
`goose session`	démarre une session interactive
`goose session start --name api-refactor`	session nommée
`goose session resume api-refactor`	reprise
`goose session list`	liste sessions
`goose run -t "tâche"`	one-shot
`goose run -i recipe.yaml`	exécution d'une recipe
`goose configure`	setup provider/modèle
`goose mcp`	gestion MCP servers
`goose update`	mise à jour
`goose info`	informations système

3.3.5 Intégration IDE

- **VSCode** : via le serveur MCP `goose-vscode`.
- **JetBrains** : via plugin Goose officiel + serveur MCP `goose-jetbrains`.
- **Neovim**, **Emacs** : intégration via terminal.
- **UI desktop** : Goose Desktop (binaire séparé).

3.3.6 Tarification

100 % gratuit, open-source. Coût = API du modèle choisi (ou gratuit avec Ollama local).

3.4 2.4 OpenHands (ex-OpenDevin)

Nom du binaire : `openhands` (CLI), GUI via Docker **Auteur** : All Hands AI (USA), précédemment communauté OpenDevin **Pays d'origine** : USA **Première sortie** : mars 2024 (sous le nom OpenDevin), renommé OpenHands en septembre 2024 **Licence** : MIT **Site officiel** : https://www.all-hands.dev **Dépôt GitHub** : https://github.com/All-Hands-AI/OpenHands

3.4.1 Description

OpenHands est une plateforme d'agents autonomes pensée pour l'autonomie complète : l'agent dispose d'un sandbox Docker dédié dans lequel il peut écrire du code, exécuter des commandes, naviguer dans un navigateur (via Playwright), gérer des fichiers, etc. Initialement projet communautaire OpenDevin (réplique open-source de Devin de Cognition Labs), le projet est désormais maintenu par All Hands AI. Il propose une UI web et un mode CLI/headless.

3.4.2 Caractéristiques

- Sandbox Docker complet (filesystem, shell, navigateur)
- Agents pluggables (CodeActAgent par défaut, PlannerAgent, BrowsingAgent…)
- Modèles via LiteLLM (Anthropic, OpenAI, Mistral, locaux…)
- Microagents (instructions par sujet, détection automatique)
- Trajectoires : enregistrement et replay des sessions
- Mode CLI (`openhands`) et mode GUI (web)
- Support GitHub Actions (mode headless)

3.4.3 Installation

Debian/Ubuntu / Fedora (tous via Docker) :

```
# Mode GUI Web
docker pull docker.all-hands.dev/all-hands-ai/runtime:latest
docker pull docker.all-hands.dev/all-hands-ai/openhands:latest

docker run -it --rm --pull=always \
    -e SANDBOX_RUNTIME_CONTAINER_IMAGE=docker.all-hands.dev/all-hands-ai/runtime:latest \
    -e LOG_ALL_EVENTS=true \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v ~/.openhands:/.openhands \
    -p 3000:3000 \
    --add-host host.docker.internal:host-gateway \
    --name openhands-app \
    docker.all-hands.dev/all-hands-ai/openhands:latest

# Mode CLI / headless via pip (avec image runtime Docker)
pipx install openhands-ai
openhands cli
```

3.4.4 Refcard

Commande	Description
`openhands cli`	CLI interactive
`openhands gui`	démarre l'UI web
`openhands serve`	API headless
`openhands -t "task"`	mode autonome
`openhands --config-file config.toml`	configuration
`openhands --replay trajectory.json`	replay

3.4.5 Intégration IDE

- **VSCode** : pas d'extension officielle, mais possibilité d'utiliser le runtime Docker comme dev container.
- **GitHub Action** : `openhands-action` pour CI.
- **Browser** : la GUI web est elle-même un IDE simplifié.

3.4.6 Tarification

Open-source gratuit. **All Hands AI Cloud** propose une version SaaS managée (payante).

3.5 2.5 Cline

Nom du binaire : `cline` (extension VSCode principalement, pas vraiment standalone CLI) **Auteur** : Saoud Rizwan / Cline Bot Inc. (USA) **Pays d'origine** : USA **Première sortie** : juillet 2024 (sous le nom *Claude Dev*, renommé Cline en septembre 2024) **Licence** : Apache 2.0 **Site officiel** : https://cline.bot **Dépôt GitHub** : https://github.com/cline/cline

3.5.1 Description

À l'origine, Cline est une **extension VSCode** très populaire (3M+ installations), mais il existe maintenant une version CLI/headless `cline` (en bêta) qui permet de lancer l'agent depuis un terminal ou en CI. Cline introduit un mode *Plan / Act* (planification puis exécution) très lisible.

3.5.2 Caractéristiques

- Multi-modèles via LiteLLM
- Plan mode / Act mode bien différenciés
- Lecteur fichier auto, exécution shell, contrôle navigateur (via VSCode webview)
- Approbation par défaut requise pour chaque action
- Support MCP
- Cli en bêta : `npx cline`

3.5.3 Installation

Extension VSCode :

```
code --install-extension saoudrizwan.claude-dev
```

CLI (bêta) :

```
sudo npm install -g @cline/cli
cline --help
```

3.5.4 Refcard (CLI bêta)

Commande	Description
`cline`	session interactive
`cline -p "prompt"`	one-shot
`cline --model claude-sonnet-4.5`	choix modèle
`cline --plan-only`	mode plan uniquement
`cline mcp list`	serveurs MCP

3.5.5 Intégration IDE

- **VSCode / VSCodium** : extension officielle (cible principale).
- **Cursor**, **Windsurf** : compatibles (étendent VSCode).
- **JetBrains** : plugin officiel « Cline for JetBrains » lancé en 2025.

3.5.6 Tarification

100 % gratuit, open-source. Coût = appels API.

3.6 2.6 Continue

Nom du binaire : `continue` (CLI), extensions IDE **Auteur** : Continue Dev, Inc. (USA) **Pays d'origine** : USA **Première sortie** : juin 2023 **Licence** : Apache 2.0 **Site officiel** : https://continue.dev **Dépôt GitHub** : https://github.com/continuedev/continue

3.6.1 Description

Continue est probablement l'**alternative open-source à Copilot la plus directe** : autocomplétion *inline* (FIM, *fill-in-the-middle*), chat latéral, edits inline, *agents customs*. Initialement extension VSCode/JetBrains, le projet propose désormais une CLI (`continue`) et un format de configuration unifié `config.yaml` ou `config.json`. Les utilisateurs peuvent partager des « assistants » via le **Continue Hub**.

3.6.2 Caractéristiques

- Autocomplete (FIM avec Codestral, Qwen, Codestral-MoE…)
- Chat avec contexte fichiers/repo
- Edits inline (Ctrl+I)
- Agents customs avec outils
- Support MCP
- Continue Hub (marketplace de configs)
- CLI pour utilisation hors-IDE
- Self-hosting friendly (Ollama, vLLM)

3.6.3 Installation

Extensions IDE :

```
code --install-extension Continue.continue
# JetBrains : via plugin marketplace, search "Continue"
```

CLI :

```
sudo npm install -g @continuedev/cli
continue --help
```

3.6.4 Refcard CLI

Commande	Description
`continue`	démarre l'agent CLI
`continue -p "prompt"`	one-shot
`continue --config ~/.continue/config.yaml`	config explicite
`continue serve`	démarre serveur local

3.6.5 Intégration IDE

- **VSCode / VSCodium** : extension officielle (très complète).
- **JetBrains** (toute la suite) : plugin officiel.
- **Neovim** : `continue.nvim` non-officiel.

3.6.6 Tarification

Open-source gratuit. **Continue Hub** offre des plans payants pour partage privé d'assistants en équipe.

3.7 2.7 Open Interpreter

Nom du binaire : `interpreter` **Auteur** : Killian Lucas / OpenInterpreter (USA) **Pays d'origine** : USA **Première sortie** : juillet 2023 **Licence** : AGPL v3 **Site officiel** : https://openinterpreter.com **Dépôt GitHub** : https://github.com/OpenInterpreter/open-interpreter

3.7.1 Description

Open Interpreter est l'un des premiers projets « ChatGPT exécute du code sur ta machine ». L'idée : le LLM écrit du code (Python, JavaScript, shell, R, AppleScript…) qui est exécuté localement, et la sortie est renvoyée au LLM. Cela en fait un *Code Interpreter local* universel (data analysis, automatisation, contrôle d'OS).

3.7.2 Caractéristiques

- Exécute Python, JS, Shell, R, AppleScript, PowerShell, HTML…
- Multi-modèles (OpenAI, Anthropic, locaux via LiteLLM)
- Mode `--os` : contrôle de la souris/clavier (Computer Use)
- Mode `--vision` : OCR de captures d'écran
- Mode `--safe` : confirme chaque exécution
- Plugin `01` pour assistant vocal

3.7.3 Installation

Debian/Ubuntu / Fedora :

```
pipx install open-interpreter
# ou
pip install --user open-interpreter
interpreter
```

Avec Docker (pour isolation) :

```
docker run -it --rm openinterpreter/openinterpreter
```

3.7.4 Refcard

Commande	Description
`interpreter`	session interactive
`interpreter --local`	utilise un modèle local (Ollama/LM Studio)
`interpreter --vision`	active vision
`interpreter --os`	mode contrôle d'OS
`interpreter --safe`	confirme chaque commande
`interpreter --auto-run`	exécute sans demander
`interpreter -y`	auto-approve
`interpreter --model claude-sonnet-4.5`	choix modèle
`interpreter --conversation_filename`	reprise

3.7.5 Intégration IDE

Pas d'intégration IDE dédiée : Open Interpreter est conçu pour le terminal et les notebooks Jupyter (`from interpreter import interpreter; interpreter.chat()`).

3.7.6 Tarification

Gratuit, open-source. Coût = API du modèle.

3.8 2.8 gpt-engineer

Nom du binaire : `gpt-engineer` (alias `gpte`) **Auteur** : Anton Osika (Suède), désormais maintenu par AntonOsika/gpt-engineer-org **Pays d'origine** : Suède **Première sortie** : juin 2023 **Licence** : MIT **Site officiel** : https://gptengineer.app **Dépôt GitHub** : https://github.com/AntonOsika/gpt-engineer

3.8.1 Description

gpt-engineer génère un projet logiciel **complet** à partir d'une simple description en langage naturel. À la différence des agents conversationnels, l'utilisateur fournit une *spec* (fichier `prompt`), et le générateur produit l'arborescence complète, les fichiers, et même des tests. Il peut aussi améliorer un projet existant en mode *improve*.

3.8.2 Caractéristiques

- Génération de projets from-scratch
- Mode `improve` (modifications sur projet existant)
- Mode `bench` (évaluation)
- Multi-modèles (OpenAI, Anthropic, Azure, locaux)
- Self-healing code (retry sur erreurs)
- Le projet a donné naissance à **Lovable** (gptengineer.app, hébergé)

3.8.3 Installation

Debian/Ubuntu / Fedora :

```
pipx install gpt-engineer
mkdir mon-projet && cd mon-projet
echo "Une app TODO en Python avec FastAPI" > prompt
export OPENAI_API_KEY=sk-...
gpte .
```

3.8.4 Refcard

Commande	Description
`gpte projects/dossier`	génère ou améliore
`gpte . --improve`	améliore le projet courant
`gpte . --use_custom_preprompts`	preprompts perso
`gpte . --model gpt-4o`	choix modèle
`gpte . --temperature 0.1`	température
`gpte --help`	aide

3.8.5 Intégration IDE

Pas d'intégration directe ; le projet généré peut être ouvert dans n'importe quel IDE.

3.8.6 Tarification

Open-source gratuit. Le service hébergé **Lovable** (anciennement gptengineer.app) est payant.

3.9 2.9 gptme

Nom du binaire : `gptme` **Auteur** : Erik Bjäreholt (Suède) **Pays d'origine** : Suède **Première sortie** : 2023 **Licence** : MIT **Site officiel** : https://gptme.org **Dépôt GitHub** : https://github.com/gptme/gptme

3.9.1 Description

gptme est un agent CLI Python générique conçu pour être un *« ChatGPT pour le terminal avec accès aux outils »*. Il dispose d'un large catalogue d'outils intégrés (shell, Python, fichiers, navigateur Playwright, RAG, vision…) et propose une logique d'agent en boucle simple. Il peut tourner *headless* en CI.

3.9.2 Caractéristiques

- Outils intégrés : shell, python, save/patch/read, browser (Playwright), screenshot, RAG, TTS
- Multi-modèles (OpenAI, Anthropic, Gemini, Groq, locaux)
- Web UI optionnelle (`gptme-server`)
- Conversations persistantes
- Mode non-interactif `gptme --non-interactive`
- Sub-agents (un agent peut en lancer un autre)

3.9.3 Installation

Debian/Ubuntu / Fedora :

```
pipx install gptme
# avec extras
pipx install 'gptme[server,browser,datascience]'
```

3.9.4 Refcard

Commande	Description
`gptme "prompt"`	one-shot
`gptme`	mode interactif
`gptme -m anthropic/claude-sonnet-4.5`	choix modèle
`gptme --tools shell,python`	restreint outils
`gptme --workspace /chemin`	workspace
`gptme --resume`	reprend session
`gptme-server`	UI web
`gptme --non-interactive "task"`	headless

3.9.5 Intégration IDE

Pas d'intégration IDE dédiée. Souvent utilisé en complément de Vim/Helix dans un panneau tmux.

3.9.6 Tarification

Open-source gratuit.

3.10 2.10 SWE-agent

Nom du binaire : `sweagent` **Auteur** : Princeton NLP (USA) **Pays d'origine** : USA **Première sortie** : avril 2024 **Licence** : MIT **Site officiel** : https://swe-agent.com **Dépôt GitHub** : https://github.com/SWE-agent/SWE-agent

3.10.1 Description

SWE-agent est issu du **benchmark SWE-bench** (Princeton) : c'est l'agent de référence pour la résolution autonome de bugs GitHub. Il dispose d'une *Agent-Computer Interface* (ACI) optimisée — un set d'outils spécifiquement conçus pour qu'un LLM les utilise efficacement (vue de fichier paginée, recherche, édition contrainte). Il a aussi un mode **EnIGMA** pour les CTF de cybersécurité.

3.10.2 Caractéristiques

- Agent computer interface (ACI) ultra-optimisée
- Sandbox Docker
- Configurable via fichiers YAML (`config/`)
- Multi-modèles (LiteLLM)
- Support de SWE-bench (évaluation reproductible)
- Mode **EnIGMA** pour CTF/sécurité offensive (autorisé)

3.10.3 Installation

Debian/Ubuntu / Fedora :

```
git clone https://github.com/SWE-agent/SWE-agent
cd SWE-agent
pipx install --editable .
# Docker requis
sweagent --help
```

3.10.4 Refcard

Commande	Description
`sweagent run`	résolution d'une issue
`sweagent run --config config/default.yaml --agent.model.name gpt-4o --env.repo.path /chemin`	issue locale
`sweagent run --env.repo.github_url URL --env.issue.github_url ISSUE_URL`	issue GitHub
`sweagent run-batch`	batch d'issues
`sweagent inspect`	UI web pour traces

3.10.5 Intégration IDE

Pas d'intégration IDE — c'est un outil de recherche / batch.

3.10.6 Tarification

Open-source gratuit.

3.11 2.11 Mentat

Nom du binaire : `mentat` **Auteur** : AbanteAI (USA) **Pays d'origine** : USA **Première sortie** : 2023 **Licence** : Apache 2.0 **Dépôt GitHub** : https://github.com/AbanteAI/mentat

3.11.1 Description

Mentat est un assistant CLI multi-fichiers pré-Aider, qui propose au LLM la pile de fichiers d'un projet et applique des modifications globales. Il propose également un mode édition collaborative (deux développeurs partagent un mentat). Le projet est moins actif depuis 2024 mais reste utilisable.

3.11.2 Caractéristiques

- Multi-fichiers, refactor large
- *Auto-context* : Mentat sélectionne tout seul les fichiers pertinents
- Multi-modèles (OpenAI, Anthropic, Azure)
- Mode collaborative
- Assertions de tests (re-run jusqu'à passage)

3.11.3 Installation

```
pipx install mentat
mentat fichier1.py dossier/
```

3.11.4 Refcard

Commande	Description
`mentat .`	session sur le projet
`mentat -i`	inclut sortie de commandes
`/include fichier`	ajoute
`/exclude fichier`	retire
`/undo`	annule
`/help`	aide

3.11.5 Intégration IDE

Pas d'intégration IDE.

3.11.6 Tarification

Open-source gratuit.

3.12 2.12 Smol Developer

Nom du binaire : `smol`, `smol-dev` **Auteur** : Swyx / smol-ai (USA) **Pays d'origine** : USA **Première sortie** : mai 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/smol-ai/developer

3.12.1 Description

Smol Developer est l'un des projets phares de la « vibe coding » de 2023 : un script de ~200 lignes qui prend une *spec* en input et génère un projet complet. Depuis, il sert surtout d'inspiration ou de base à des projets plus aboutis (gpt-engineer, Lovable…).

3.12.2 Caractéristiques

- Script unique très court
- Génération from-scratch
- Multi-fichiers
- Pas vraiment maintenu activement

3.12.3 Installation

```
git clone https://github.com/smol-ai/developer
cd developer
pip install -r requirements.txt
modal token new   # ou export OPENAI_API_KEY
python main.py "ton prompt"
```

3.12.4 Refcard

Commande	Description
`python main.py "spec"`	génère un projet
`python debugger.py "erreur"`	mode debug

3.12.5 Tarification

Gratuit.

3.13 2.13 Charm Crush

Nom du binaire : `crush` **Auteur** : Charmbracelet (USA) **Pays d'origine** : USA **Première sortie** : 2025 (successeur de l'utilitaire `mods` côté agentique) **Licence** : MIT **Dépôt GitHub** : https://github.com/charmbracelet/crush

3.13.1 Description

Crush est l'agent CLI agentique de Charmbracelet, lancé en 2025, qui complète l'écosystème *Charm* (mods, glow, gum, bubbletea). Là où `mods` reste un filtre LLM, Crush propose **outils, sessions, hooks et MCP** dans une UI TUI très soignée typique de Charm.

3.13.2 Caractéristiques

- TUI léchée (Bubble Tea)
- Multi-modèles
- Support MCP
- Sessions persistantes
- Outils intégrés (filesystem, shell, git)
- Modes plan/auto

3.13.3 Installation

Debian/Ubuntu (dépôt Charm) :

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://repo.charm.sh/apt/gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/charm.gpg
echo "deb [signed-by=/etc/apt/keyrings/charm.gpg] https://repo.charm.sh/apt/ * *" | sudo tee /etc/apt/sour
sudo apt update && sudo apt install crush
```

Fedora :

```
echo '[charm]
name=Charm
baseurl=https://repo.charm.sh/yum/
enabled=1
gpgcheck=1
gpgkey=https://repo.charm.sh/yum/gpg.key' | sudo tee /etc/yum.repos.d/charm.repo
sudo dnf install crush
```

Go :

```
go install github.com/charmbracelet/crush@latest
```

3.13.4 Refcard

Commande	Description
`crush`	TUI interactive
`crush -p "prompt"`	one-shot
`crush sessions`	liste sessions
`crush mcp`	gestion MCP
`crush --model claude-sonnet-4.5`	choix modèle

3.13.5 Intégration IDE

Pas d'intégration IDE dédiée ; utilisable côte-à-côte avec un IDE.

3.13.6 Tarification

Open-source gratuit.

3.14 2.14 Forge / Code Forge

Nom du binaire : `forge` **Auteur** : Antinomy (Inde / USA) **Première sortie** : 2024 **Licence** : Apache 2.0 (à vérifier) **Dépôt GitHub** : https://github.com/antinomyhq/forge

3.14.1 Description

Code Forge (aussi appelé simplement *Forge*) est un agent CLI Rust pour le développement assisté. Il se positionne comme alternative légère à Aider/Plandex avec une UI TUI Bubble Tea-like et un support multi-modèles. Le projet est plus jeune mais en progression rapide.

3.14.2 Caractéristiques

- Rust, binaire unique
- Multi-modèles
- TUI moderne
- Sessions
- Support MCP

3.14.3 Installation

```
# Debian/Ubuntu/Fedora
curl -L https://raw.githubusercontent.com/antinomyhq/forge/main/install.sh | bash
# ou
cargo install forge
```

3.14.4 Refcard

Commande	Description
`forge`	session interactive
`forge -p "prompt"`	one-shot
`forge config`	configuration

3.14.5 Tarification

Open-source gratuit.

3.15 2.15 Roo Code

Nom du binaire : extension VSCode principalement, CLI via `roo-cli` en bêta **Auteur** : Roo Vet et al. (USA) **Première sortie** : 2024 (fork de Cline) **Licence** : Apache 2.0 **Dépôt GitHub** : https://github.com/RooVetGit/Roo-Code

3.15.1 Description

Roo Code est un fork plus avancé de Cline qui ajoute des **modes** spécialisés (Code, Architect, Ask, Debug, Custom) et des fonctionnalités supplémentaires : *boomerang tasks* (sous-tâches), *Browser Use*, *MCP marketplace*, *checkpoints*.

3.15.2 Caractéristiques

- Modes personnalisables avec prompts spécifiques
- Sous-tâches récursives
- Marketplace MCP intégré
- Browser Use (via Playwright)
- Checkpoints (snapshots du repo)

3.15.3 Installation

Extension VSCode :

```
code --install-extension RooVeterinaryInc.roo-cline
```

CLI bêta :

```
sudo npm install -g @roo-code/cli
```

3.15.4 Tarification

Open-source gratuit.

3.16 2.16 Autres mentions notables

- **Devin** (Cognition Labs, USA) — agent autonome SaaS, pas de CLI publique mais une API. https://www.cognition.ai
- **Devon** (entropy-research/Devon) — agent open-source inspiré de Devin, MIT.
- **GPT Pilot / Pythagora** (Pythagora-io/gpt-pilot) — agent autonome qui pose des questions de clarification avant de coder. https://github.com/Pythagora-io/gpt-pilot
- **Sweep AI** (sweepai/sweep) — agent qui ouvre des PRs sur GitHub à partir d'issues.
- **Tabby** (TabbyML/tabby) — alternative self-hosted à Copilot, pas de CLI agentique mais un serveur d'inférence + plugins IDE. https://github.com/TabbyML/tabby
- **Codeium / Windsurf** — orienté IDE, mais Cascade dispose d'un mode terminal.
- **Bloop** (BloopAI/bloop) — recherche sémantique de code en CLI/desktop.
- **Khoj** (khoj-ai/khoj) — assistant personnel CLI/chat, RAG sur notes/markdown/email.

Chapter 4

Partie III — CLI LLM génériques

Cette partie regroupe les outils CLI destinés à interagir directement avec un LLM (cloud ou local), souvent avec une orientation *pipe-friendly* pour s'intégrer aux flux Unix. Ils ne sont pas des agents autonomes mais des **filtres LLM** qu'on peut chaîner avec `cat`, `grep`, `jq`, `git diff`, etc.

4.1 3.1 llm (Simon Willison)

Nom du binaire : `llm` **Auteur** : Simon Willison (Royaume-Uni / USA), co-créateur de Django **Pays d'origine** : Royaume-Uni / USA **Première sortie** : avril 2023 **Licence** : Apache 2.0 **Site officiel** : https://llm.datasette.io **Dépôt GitHub** : https://github.com/simonw/llm

4.1.1 Description

`llm` est l'un des CLI LLM les plus matures de l'écosystème Unix. Conçu par Simon Willison, il joue le rôle de couteau suisse pour interagir avec n'importe quel modèle (cloud ou local) via une interface uniforme. Sa force tient à son **système de plugins** très riche (un plugin par fournisseur ou famille de modèles) et à sa **base SQLite** qui historise toutes les conversations pour les rendre interrogeables.

4.1.2 Caractéristiques

- Pipes shell natifs (entrée stdin)
- Historique SQLite (`~/.config/io.datasette.llm/logs.db`)
- Templates de prompts (`llm templates`)
- Multi-modèles via plugins (`llm-anthropic`, `llm-gemini`, `llm-ollama`, `llm-mlx`, `llm-gpt4all`, `llm-llamafile`…)
- Embeddings et RAG via `llm embed`, `llm similar`
- Fragments réutilisables, attachments multimodaux (images, PDF, audio)
- Tools / function calling
- Plugins MCP en cours d'intégration

4.1.3 Installation

```
# Debian/Ubuntu
sudo apt install -y pipx && pipx ensurepath
pipx install llm

# Fedora
sudo dnf install -y pipx && pipx ensurepath
pipx install llm
```

```
# macOS / Linuxbrew
brew install llm

# pip standard
pip install --user llm
```

4.1.4 Refcard

```
llm "Explique le théorème de Bayes"
llm -m gpt-4o "résume ce fichier" < rapport.txt
cat code.py | llm -s "Trouve les bugs"
llm chat -m claude-sonnet-4.5
llm keys set openai
llm models                        # liste des modèles
llm install llm-anthropic         # installer plugin
llm logs -n 5                     # 5 derniers échanges
llm logs -q "docker"              # recherche full-text
llm templates edit summarize
llm -t summarize < article.md
llm embed-multi docs -m 3-small --files docs '**/*.md' --store
llm similar docs -c "déploiement Kubernetes"
llm -a image.jpg "que vois-tu ?"
```

4.1.5 Backends

OpenAI (natif), Anthropic, Gemini, Mistral, Cohere, Ollama, llama.cpp, MLX, GPT4All, Groq, OpenRouter, Together, etc.

4.1.6 Intégration IDE/shell

- Complétion shell : `llm install llm-cmd`
- Pipe-friendly : utilisable dans Makefiles, hooks Git, scripts CI
- Editor integration via `llm edit` (édite la dernière réponse dans `$EDITOR`)

4.1.7 Cas d'usage

Génération de messages de commit, explication de stack-traces, RAG documentaire local, expérimentation rapide multi-modèles, base SQLite analysable avec `datasette`.

4.2 3.2 ShellGPT (sgpt)

Nom du binaire : `sgpt` **Auteur** : Farkhod Sadykov (TheR1D), Ouzbékistan / Allemagne **Pays d'origine** : Ouzbékistan / Allemagne **Première sortie** : décembre 2022 **Licence** : MIT **Dépôt GitHub** : https://github.com/TheR1D/shell_gpt

4.2.1 Description

ShellGPT cible explicitement l'usage shell : générer des commandes, du code, ou des réponses Markdown directement dans le terminal. Il introduit la notion de **rôles** (system prompts réutilisables) et un **mode REPL** persistant. Il propose un **mode shell** qui propose, exécute ou décrit la commande générée. Il fonctionne avec n'importe quel endpoint compatible OpenAI, ce qui le rend compatible avec Ollama, LM Studio, vLLM, etc.

4.2.2 Caractéristiques

- Pipes shell natifs
- Historique des chats (`--chat <name>`)
- Rôles personnalisés (`--role`)
- Multi-modèles (OpenAI + tout serveur compatible OpenAI)
- Function calling
- Intégrations clavier zsh/bash/fish (`Ctrl+L`)

4.2.3 Installation

```
# Debian/Ubuntu / Fedora
pipx install shell-gpt
# ou
pip install --user shell-gpt
```

4.2.4 Refcard

```
sgpt "Quelle est la capitale du Pérou ?"
sgpt --shell "trouver tous les .log > 100 Mo"
sgpt --code "fonction Python pour parser /etc/passwd"
sgpt --describe-shell "tar -czvf"
sgpt --chat dev "écris une fonction de tri"
sgpt --repl temp
sgpt --role create coder
sgpt --role coder "écris un Dockerfile"
cat README.md | sgpt "résume en 5 bullets"
sgpt --functions "quel temps fait-il à Paris ?"
```

4.2.5 Backend Ollama

```
export DEFAULT_MODEL="ollama/llama3.1"
export API_BASE_URL="http://localhost:11434/v1"
```

4.2.6 Intégration shell

Installation d'une keybinding via `_sgpt_zsh` (Ctrl+L pour transformer la ligne de commande en prompt).

4.3 3.3 aichat

Nom du binaire : `aichat` **Auteur** : sigoden (Chine) **Pays d'origine** : Chine **Première sortie** : mars 2023 **Licence** : MIT OR Apache-2.0 **Dépôt GitHub** : https://github.com/sigoden/aichat

4.3.1 Description

`aichat` est probablement le CLI LLM **le plus complet en termes de fonctionnalités intégrées** : chat, REPL, RAG, agents, function calling, serveur HTTP, WebUI embarquée — le tout dans un binaire Rust unique. Il supporte plus de 20 fournisseurs LLM via une **configuration YAML** unifiée.

4.3.2 Caractéristiques

- Pipes shell + mode shell (`-e`)
- Sessions multiples persistées
- Rôles (system prompts), macros
- RAG intégré (vector store + recherche)
- Function calling et **agents** scriptables
- Serveur HTTP compatible OpenAI (`aichat --serve`)
- WebUI / Playground / Arena intégrés
- Multi-modèles (20+ providers)

4.3.3 Installation

```
# Debian/Ubuntu (binaire)
curl -fsSL https://github.com/sigoden/aichat/releases/latest/download/aichat-x86_64-unknown-linux-gnu.tar.g
  | sudo tar -xz -C /usr/local/bin aichat

# Cargo (Debian/Fedora)
cargo install aichat

# Snap
sudo snap install aichat
```

4.3.4 Refcard

```
aichat "Explique HTTP/3"
aichat -m openai:gpt-4o "résume" < rapport.md
aichat -e "compresser tout le dossier en tar.gz"      # exécution shell
aichat -c "trouver les doublons en bash"               # code only
aichat --session debug                                 # session persistée
aichat --role coder "écris un test pytest"
aichat --rag mes-docs                                  # RAG store
aichat --agent translator "traduis en italien"
aichat --serve 0.0.0.0:8080                            # API + WebUI
aichat --list-models
```

4.3.5 Backends

OpenAI, Anthropic, Gemini, Mistral, Cohere, Ollama, vLLM, Groq, DeepSeek, OpenRouter, Bedrock, Vertex, Azure, Cloudflare Workers AI, Replicate, etc.

4.3.6 Intégrations

Completion zsh/bash/fish (`aichat --completion zsh`), keybindings Ctrl+O.

4.4 3.4 mods (Charmbracelet)

Nom du binaire : `mods` **Auteur** : Charmbracelet (USA), équipe derrière `glow`, `gum`, `bubbletea` **Pays d'origine** : USA **Première sortie** : juin 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/charmbracelet/mods

4.4.1 Description

`mods` est l'AI CLI de Charmbracelet, conçu pour s'intégrer parfaitement à un workflow Unix : il lit `stdin`, écrit du Markdown coloré (rendu Glamour), et sait formater le résultat (code, JSON, markdown). Pensé comme un **filtre shell** plutôt qu'un chatbot.

4.4.2 Caractéristiques

- Pipes shell first-class
- Historique des conversations (`mods -l`, `mods -s`)
- Templates / rôles via `mods.yml`
- Multi-modèles (OpenAI, Anthropic, Cohere, Groq, Ollama, Azure, Perplexity, Google)
- Glamour rendering (Markdown coloré)

4.4.3 Installation

Debian/Ubuntu (dépôt Charm) :

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://repo.charm.sh/apt/gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/charm.gpg
echo "deb [signed-by=/etc/apt/keyrings/charm.gpg] https://repo.charm.sh/apt/ * *" | sudo tee /etc/apt/sour
sudo apt update && sudo apt install mods
```

Fedora :

```
echo '[charm]
name=Charm
baseurl=https://repo.charm.sh/yum/
enabled=1
gpgcheck=1
gpgkey=https://repo.charm.sh/yum/gpg.key' | sudo tee /etc/yum.repos.d/charm.repo
sudo dnf install mods
```

Go :

```
go install github.com/charmbracelet/mods@latest
```

4.4.4 Refcard

```
mods "qu'est-ce qu'un mutex ?"
cat main.go | mods "trouve les bugs"
git diff | mods "écris un message de commit conventionnel"
mods -m gpt-4o "réécris ce paragraphe"
mods -f                          # format markdown
mods -r                          # raw output
mods --role coder "écris un test"
mods -P                          # liste de prompts
mods -l                          # liste les conversations
mods -s "perf-debug"             # sauvegarde
mods -c "perf-debug"             # continue
mods --settings                  # éditeur de config
```

4.4.5 Cas d'usage emblématiques

```
git diff | mods "écris un commit conventionnel"
dmesg | mods "explique les erreurs"
kubectl get pods -o yaml | mods "diagnostique"
ps auxf | mods "quels process suspect ?"
```

4.5 3.5 tgpt

Nom du binaire : `tgpt` **Auteur** : Andrew (aandrew-me), Bangladesh **Pays d'origine** : Bangladesh **Première sortie** : 2023 **Licence** : GPL-3.0 **Dépôt GitHub** : https://github.com/aandrew-me/tgpt

4.5.1 Description

`tgpt` (Terminal GPT) se distingue par son **utilisation sans clé API** : il interroge des fournisseurs publics gratuits (Phind, Blackbox, KoboldAI, DuckDuckGo AI, etc.) tout en restant compatible avec n'importe quelle API OpenAI-like ou Ollama. Idéal pour usage personnel sans inscription ou pour démos.

4.5.2 Caractéristiques

- Pas de clé API requise par défaut
- Mode interactif et multiline
- Mode shell, code, image
- Historique optionnel
- Multi-providers via `--provider`
- Génération d'images (Pollinations, Arta)

4.5.3 Installation

```
# Debian/Ubuntu/Fedora (script officiel)
curl -sSL https://raw.githubusercontent.com/aandrew-me/tgpt/main/install | bash -s /usr/local/bin

# Snap
sudo snap install tgpt

# Brew
brew install tgpt

# Go
go install github.com/aandrew-me/tgpt/v2@latest
```

4.5.4 Refcard

```
tgpt "Comment marche TCP ?"
tgpt -s "convertir tous les .png en webp"   # shell command
tgpt -c "fonction Python fibonacci"          # code only
tgpt -i                                      # interactive
tgpt -m                                      # multiline
tgpt --img "un chat astronaute"              # génération image
tgpt --provider phind "..."
tgpt --provider openai --model gpt-4o-mini --key sk-... \
  --url https://api.openai.com/v1/chat/completions "..."
echo "log..." | tgpt "explique cette erreur"
```

4.6 3.6 fabric

Nom du binaire : `fabric` (parfois `fabric-ai`) **Auteur** : Daniel Miessler (USA), expert sécurité offensive **Pays d'origine** : USA **Première sortie** : janvier 2024 **Licence** : MIT **Dépôt GitHub** : https://github.com/danielmiessler/fabric

4.6.1 Description

Fabric est **un framework de prompts modulaires** plus qu'un simple chat CLI. Il propose plus de 200 **patterns** (system prompts soigneusement écrits) couvrant analyse de code, OSINT, résumé, écriture, *threat modeling*, etc. L'idée : `<input> | fabric -p <pattern>` pour appliquer une recette d'IA reproductible.

4.6.2 Caractéristiques

- Pipes shell first-class
- Catalogue de **patterns** versionnés (`fabric -L`)
- Multi-modèles (OpenAI, Anthropic, Ollama, Gemini, Groq, Azure, etc.)
- Sessions, contextes, variables
- Streaming, sortie YAML/JSON
- Serveur HTTP / API REST

4.6.3 Installation

```
# Debian/Ubuntu/Fedora
go install github.com/danielmiessler/fabric@latest
# binaire dans ~/go/bin/fabric

fabric --setup                  # config interactive
fabric -U                       # télécharge les patterns
```

4.6.4 Refcard

```
fabric -L                               # liste patterns
fabric -p summarize < article.txt
fabric -p extract_wisdom -m gpt-4o
fabric -p write_essay --stream
yt-dlp --get-id 'URL' | fabric -y "URL" -p extract_wisdom
fabric -p analyze_threat_report < rapport.pdf.txt
fabric --listmodels
fabric --serve                          # API REST
fabric --session pentest -p ...
fabric --context corp -p ...
```

4.6.5 Cas d'usage

Analyse de threat reports, extraction de *wisdom* depuis vidéos YouTube, génération d'essais, code review automatisé, OSINT.

4.7 3.7 gptscript

Nom du binaire : `gptscript` **Auteur** : Acorn Labs (USA) **Pays d'origine** : USA **Première sortie** : mars 2024 **Licence** : Apache-2.0 **Dépôt GitHub** : https://github.com/gptscript-ai/gptscript

4.7.1 Description

`gptscript` introduit un **langage déclaratif** (fichiers `.gpt`) pour décrire des agents : tools, sub-agents, fichiers de contexte, instructions en langage naturel. Le runtime exécute le script en orchestrant les appels LLM et les outils (HTTP, scripts shell, OpenAPI, MCP).

4.7.2 Caractéristiques

- DSL `.gpt` natif
- Tools : commandes shell, OpenAPI, Python, Node, MCP
- Workflow agent multi-étapes
- Multi-modèles (OpenAI, Anthropic, Azure, local)
- Mode chat ou one-shot
- Confirmations interactives pour outils dangereux

4.7.3 Installation

```
# Debian/Ubuntu/Fedora
curl https://get.gptscript.ai/install.sh | sh

# Brew
brew install gptscript-ai/tap/gptscript
```

4.7.4 Refcard

```
gptscript hello.gpt
gptscript --chat hello.gpt
gptscript --confirm github.com/gptscript-ai/code-review
gptscript --debug --quiet=false agent.gpt --topic "k8s"
gptscript --workspace ./ws agent.gpt
gptscript --default-model gpt-4o agent.gpt
```

Exemple `hello.gpt` :

```
tools: sys.read, sys.write
description: lit un fichier et écrit un résumé

Lis README.md et écris un résumé dans SUMMARY.md.
```

> Note : depuis 2025, l'équipe se concentre sur **Obot/Otto** (plateforme agent). `gptscript` reste maintenu mais son successeur est à surveiller.

4.8 3.8 chatgpt-cli (kardolus)

Nom du binaire : `chatgpt` **Auteur** : Joost Kardolus (Pays-Bas) **Pays d'origine** : Pays-Bas **Licence** : MIT **Dépôt GitHub** : https://github.com/kardolus/chatgpt-cli

4.8.1 Description

Client CLI Go simple, focalisé sur la **conversation persistante** et l'usage scripté. Léger, sans dépendances Python, support OpenAI / Azure / Perplexity / tout endpoint compatible OpenAI.

4.8.2 Installation

```
# Debian/Ubuntu/Fedora
curl -L -o chatgpt https://github.com/kardolus/chatgpt-cli/releases/latest/download/chatgpt-linux-amd64
chmod +x chatgpt && sudo mv chatgpt /usr/local/bin/

# Brew
brew tap kardolus/chatgpt-cli
brew install chatgpt-cli
```

4.8.3 Refcard

```
chatgpt "Explique le pattern observer"
echo "log..." | chatgpt
chatgpt --query "réponse courte"
chatgpt --interactive
chatgpt -n "session-debug" "..."        # nouvelle conv
chatgpt --clear-history
chatgpt --list-models
chatgpt --set-model gpt-4o
```

4.9 3.9 gpt-cli (kharvd)

Nom du binaire : `gpt` **Auteur** : Daniil Kharkov (kharvd) **Licence** : MIT **Dépôt GitHub** : https://github.com/kharvd/gpt-cli

4.9.1 Description

Client conversationnel Python supportant nativement OpenAI, Anthropic, Google, Cohere et compatibles. Accent sur les **assistants paramétrables** (fichiers YAML) et un REPL ergonomique avec markdown rendu en temps réel.

4.9.2 Installation

```
pipx install gpt-command-line
```

4.9.3 Refcard

```
gpt                                    # REPL
gpt --model gpt-4o
gpt general "résume ce texte" < f.txt
gpt code "génère un script bash"
gpt dev                                # assistant custom YAML
gpt --temperature 0.2
```

4.10 3.10 smartcat (sc)

Nom du binaire : `sc` **Auteur** : Emilien Fugier (France) **Pays d'origine** : France **Licence** : Apache-2.0 **Dépôt GitHub** : https://github.com/efugier/smartcat

4.10.1 Description

`smartcat` (alias `sc`) se positionne comme **un `cat` augmenté par LLM** : pipe-friendly à 100 %, conçu pour s'intégrer dans des éditeurs (Vim/Helix/Neovim) et des chaînes shell. Le binaire est minimaliste et pensé pour la **substitution in-place** de texte.

4.10.2 Installation

```
# Debian/Ubuntu/Fedora
cargo install smartcat
# Brew
brew install smartcat
```

4.10.3 Refcard

```
sc "écris un haïku sur Rust"
cat code.py | sc "ajoute des docstrings" > code_doc.py
sc -p commit "$(git diff --staged)"
sc -e "continue cette conversation"
sc -c context.txt "résume"
echo "bonjour" | sc -p translate-fr-en
```

4.10.4 Intégration éditeur

- **Vim** : `:%!sc "refactor"`
- **Helix** : `|sc`
- Idéal pour transformer une sélection en pipeline.

4.11 3.11 chatblade

Nom du binaire : `chatblade` **Auteur** : Niels van Putten (Pays-Bas) **Licence** : MIT **Dépôt GitHub** : https://github.com/npiv/chatblade

4.11.1 Description

Outil ChatGPT pour terminal Unix. Met l'accent sur l'extraction de données (JSON, code, markdown) à partir des réponses, et sur la persistance des sessions. Très proche du flux Unix : `cat | chatblade | jq`.

4.11.2 Installation

```
pipx install chatblade
```

4.11.3 Refcard

```
chatblade "Quelle est la racine carrée de 144 ?"
echo "code..." | chatblade -e "trouve les bugs"
chatblade -p code-review < diff.patch
chatblade -l                              # last response
chatblade -r                              # raw markdown
chatblade -c 4                            # GPT-4
chatblade -t                              # token stats
chatblade -S session1 "..."               # save session
chatblade --extract code                  # extrait code blocks
chatblade --extract json
```

4.12 3.12 gorilla-cli

Nom du binaire : `gorilla` **Auteur** : Berkeley AI Research (Shishir Patil et al.), USA **Pays d'origine** : USA **Licence** : Apache 2.0 **Dépôt GitHub** : https://github.com/gorilla-llm/gorilla-cli **Site** : https://gorilla.cs.berkeley.edu

4.12.1 Description

Issu du projet de recherche **Gorilla LLM** (Berkeley), `gorilla-cli` traduit du langage naturel en commandes shell pour 1500+ APIs CLI (`gcloud`, `kubectl`, `aws`, `git`…). Il **propose plusieurs candidats** que l'utilisateur sélectionne avant exécution, ce qui en fait un outil orienté sécurité.

4.12.2 Installation

```
pipx install gorilla-cli
```

4.12.3 Refcard

```
gorilla "liste tous les pods en erreur"
gorilla "crée un bucket S3 chiffré"
gorilla "git rebase interactif des 3 derniers commits"
gorilla -- history                    # historique
```

4.13 3.13 shai (OVHcloud)

Nom du binaire : `shai` **Auteur** : OVHcloud (France) **Pays d'origine** : France **Licence** : Apache 2.0 **Dépôt GitHub** : https://github.com/ovh/shai

4.13.1 Description

Assistant shell open-source publié par **OVHcloud** (France), écrit en Rust, utilisant les modèles servis par les **AI Endpoints OVHcloud**. Pertinent pour les organisations européennes recherchant la souveraineté numérique. Compatible avec d'autres providers OpenAI-compatibles.

4.13.2 Installation

```
# Cargo
cargo install shai
# ou release binaire
curl -fsSL https://github.com/ovh/shai/releases/latest/download/shai-installer.sh | sh
```

4.13.3 Refcard

```
shai "trouve les fichiers > 100 Mo modifiés cette semaine"
shai --explain "tar -czvf archive.tgz dir"
shai --provider openai --model llama-3.3-70b
shai chat
```

4.13.4 Backends

OVH AI Endpoints (par défaut), OpenAI, Anthropic, Ollama (via configuration).

4.14 3.14 wrangler ai (Cloudflare)

Nom du binaire : `wrangler` (sous-commande `ai`) **Auteur** : Cloudflare (USA) **Licence** : MIT OR Apache-2.0 **Dépôt GitHub** : https://github.com/cloudflare/workers-sdk

4.14.1 Description

Bien que `wrangler` soit avant tout l'outil de déploiement de Cloudflare Workers, sa sous-commande `ai` permet d'exécuter directement les modèles disponibles sur **Workers AI** depuis le terminal — pratique pour des essais rapides ou des tests CI.

4.14.2 Installation

```
sudo npm install -g wrangler
# ou
pnpm add -g wrangler
wrangler login
```

4.14.3 Refcard

```
wrangler ai models                                          # liste modèles
wrangler ai run @cf/meta/llama-3.1-8b-instruct --prompt "Bonjour"
wrangler ai run @cf/black-forest-labs/flux-1-schnell --prompt "chat astronaute"
```

4.15 3.15 Autres clients CLI génériques

- **Khoj CLI** (`khoj`) — assistant personnel RAG (markdown, PDF, email). https://github.com/khoj-ai/khoj
- **AnythingLLM** — application desktop avec API CLI. https://github.com/Mintplex-Labs/anything-llm
- **NextChat CLI** (anciennement ChatGPT-Next-Web) — variante CLI peu officielle.
- **OpenCommit CLI** — voir Partie V (outil spécialisé commits).
- **HuggingFace Hub CLI** (`huggingface-cli`) — gestion de modèles, mais aussi `chat-cli` pour discuter avec des modèles HF.
- **Ollama** — bien qu'étant un runner local, sa CLI inclut un mode chat — voir Partie IV.

4.15.1 Synthèse comparative — CLI génériques

Outil	Langage	Pipes	Sessions	RAG	Tools	Multi-providers
llm	Python	Oui	Oui (SQLite)	Oui (embed)	Oui (plugins)	Oui (plugins)
sgpt	Python	Oui	Oui	Non	Oui	OpenAI-compat
aichat	Rust	Oui	Oui	Oui natif	Oui agents	Oui 20+
mods	Go	Oui	Oui	Non	limité	Oui 9+
tgpt	Go	Oui	optionnel	Non	Non	free + OAI-compat
fabric	Go	Oui	Oui	Non	patterns	Oui
gptscript	Go	partiel	Oui	Non	DSL	Oui
chatgpt-cli	Go	Oui	Oui	Non	Non	OAI-compat
gpt-cli	Python	partiel	Oui	Non	limité	Oui
smartcat	Rust	Oui	Oui	Non	Non	Oui
chatblade	Python	Oui	Oui	Non	Non	OAI-compat
gorilla-cli	Python	Non	Non	Non	Non	Berkeley/OAI
shai (OVH)	Rust	Oui	Oui	Non	limité	Oui
wrangler ai	TS/Node	partiel	Non	Non	Non	Cloudflare

Chapter 5

Partie IV — Runners de modèles LLM locaux

Cette partie couvre les outils CLI permettant **d'exécuter des modèles LLM directement sur sa machine**, sans dépendre du cloud. C'est la voie privilégiée pour la souveraineté, la confidentialité (données sensibles, code propriétaire), le travail offline, et la maîtrise des coûts. La plupart de ces outils exposent une API compatible OpenAI, ce qui permet de les utiliser comme backend pour Aider, Continue, aichat, etc.

5.1 4.1 Ollama

Nom du binaire : `ollama` **Auteur** : Ollama Inc. (USA) **Pays d'origine** : USA **Première sortie** : juillet 2023 **Licence** : MIT **Site officiel** : https://ollama.com **Dépôt GitHub** : https://github.com/ollama/ollama

5.1.1 Description

Ollama est devenu le **runner local de référence** pour LLM. Il combine une **bibliothèque de modèles** prêts à l'emploi (Llama, Mistral, Qwen, Gemma, Phi, DeepSeek, et beaucoup d'autres), une **API REST compatible OpenAI**, une CLI ergonomique inspirée de Docker (`pull`, `run`, `ps`, `rm`), et un système de **Modelfiles** pour personnaliser des modèles. Sous le capot, il s'appuie sur **llama.cpp** mais cache toute la complexité (quantization, contextes, GPU).

5.1.2 Caractéristiques

- Bibliothèque de modèles GGUF préconfigurés (registre https://ollama.com/library)
- API HTTP compatible OpenAI (`http://localhost:11434/v1`)
- Support **CUDA** (NVIDIA), **ROCm** (AMD), **Metal** (macOS), CPU
- Quantization automatique (Q4_0, Q4_K_M, Q5_K_M, Q8_0…)
- Modelfile : `FROM llama3 / SYSTEM ... / PARAMETER ...`
- Multi-utilisateurs, multi-modèles concurrents
- Outils intégrés : tools/function calling depuis 0.4
- Vision (modèles multimodaux : llava, llama3.2-vision…)
- Embeddings (`ollama embed`)
- Hub officiel + push/pull de modèles personnels

5.1.3 Installation

Debian/Ubuntu :

```
curl -fsSL https://ollama.com/install.sh | sh
# Service systemd installé automatiquement
sudo systemctl enable --now ollama
```

Fedora :

```
curl -fsSL https://ollama.com/install.sh | sh
# ou via copr
sudo dnf copr enable luminoso/ollama
sudo dnf install ollama
```

Docker / Podman :

```
docker run -d -v ollama:/root/.ollama \
  -p 11434:11434 --gpus all \
  --name ollama ollama/ollama
```

5.1.4 Refcard

```
ollama pull llama3.3                # télécharge un modèle
ollama list                         # liste local
ollama run llama3.3                 # chat interactif
ollama run llama3.3 "Bonjour"       # one-shot
echo "résume ceci..." | ollama run llama3.3
ollama ps                           # modèles en mémoire
ollama stop llama3.3                # libère VRAM
ollama rm llama3.3                  # supprime modèle
ollama show llama3.3                # détails
ollama serve                        # démarre serveur (déjà fait par systemd)
ollama create my-model -f Modelfile # crée modèle perso
ollama push myuser/my-model         # publie sur registre
ollama cp llama3.3 mon-llama        # duplique
ollama -v                           # version
```

Exemple `Modelfile` :

```
FROM llama3.3
PARAMETER temperature 0.3
PARAMETER num_ctx 8192
SYSTEM Tu es un expert Linux qui répond en français.
```

5.1.5 API OpenAI-compatible

```
curl http://localhost:11434/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.3",
    "messages": [{"role":"user","content":"Bonjour"}]
  }'
```

5.1.6 Intégration IDE

- **VSCode / VSCodium** : extensions **Continue**, **Cline**, **Ollama Autocoder**.
- **JetBrains** : plugins Continue, ProxyAI, CodeGPT.
- **Neovim** : `gen.nvim`, `ollero.nvim`, `codecompanion.nvim`.
- **Emacs** : `ellama`, `gptel`.
- **Aider, llm, sgpt, aichat, mods** : utilisable comme backend.

5.1.7 Performance

- Modèles 7B Q4 : ~5 Go VRAM, fonctionnent même sur GPU 6 Go.
- Modèles 70B Q4 : ~40 Go VRAM, nécessitent A100/H100 ou multi-GPU.

- CPU-only : possible mais lent (1-5 tok/s sur Ryzen 7 récent).

5.2 4.2 llama.cpp

Nom du binaire : `llama-cli`, `llama-server`, `llama-bench`, `llama-quantize`… **Auteur** : Georgi Gerganov + communauté **Pays d'origine** : Bulgarie + international **Première sortie** : mars 2023 **Licence** : MIT **Site officiel** : https://github.com/ggml-org/llama.cpp **Dépôt GitHub** : https://github.com/ggml-org/llama.cpp

5.2.1 Description

`llama.cpp` est l'**implémentation de référence** de l'inférence LLM en C/C++ optimisée pour CPU et GPU, sans dépendance à Python ou PyTorch. C'est la fondation sur laquelle reposent Ollama, llamafile, LM Studio, Jan, KoboldCpp, etc. Pour les utilisateurs avancés, l'utilisation directe permet un contrôle fin des paramètres et une absence totale d'overhead.

5.2.2 Caractéristiques

- C/C++ pur (Vulkan, CUDA, ROCm, Metal, BLAS, OpenCL, SYCL)
- Format **GGUF** (lui aussi natif au projet)
- Quantization 1.5 bit à 8 bit (IQ1_S, IQ2_XXS, Q4_K_M, Q8_0…)
- Outils inclus : conversion HF→GGUF, quantization, fine-tuning LoRA
- Serveur HTTP avec UI web et API OpenAI
- Support multi-modèles (parallel slots)
- *Speculative decoding*, *grammar-constrained generation*
- Multimodal : LLaVA, MiniCPM-V, Granite Vision…

5.2.3 Installation

Debian/Ubuntu (build from source) :

```
sudo apt install -y build-essential cmake git libcurl4-openssl-dev
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
cmake -B build -DGGML_CUDA=ON   # ou -DGGML_VULKAN=ON, -DGGML_HIPBLAS=ON
cmake --build build --config Release -j

# Binaires dans ./build/bin/
sudo cp build/bin/llama-* /usr/local/bin/
```

Fedora :

```
sudo dnf install -y gcc-c++ cmake git libcurl-devel cuda-toolkit-12-6
# Idem ci-dessus
```

Pré-compilé :

```
# Releases GitHub : binaires Linux x86_64, ARM64
wget https://github.com/ggml-org/llama.cpp/releases/latest/download/llama-bXXXX-bin-ubuntu-x64.zip
unzip llama-*.zip
```

5.2.4 Refcard

```
# Téléchargement de modèle (ex. via huggingface-cli ou curl)
huggingface-cli download bartowski/Meta-Llama-3.1-8B-Instruct-GGUF \
  Meta-Llama-3.1-8B-Instruct-Q4_K_M.gguf --local-dir ~/models

# Inférence one-shot
llama-cli -m ~/models/Meta-Llama-3.1-8B-Instruct-Q4_K_M.gguf \
  -p "Explique TCP en 5 phrases" -n 256

# Mode interactif
```

```
llama-cli -m model.gguf --interactive --color --reverse-prompt "User:"

# Serveur HTTP (compatible OpenAI)
llama-server -m model.gguf -c 8192 --host 0.0.0.0 --port 8080 -ngl 35

# Benchmark
llama-bench -m model.gguf -p 512 -n 128 -ngl 99

# Conversion HF → GGUF
python convert_hf_to_gguf.py /chemin/repo-hf

# Quantization
llama-quantize model-f16.gguf model-q4km.gguf Q4_K_M

# Options clés :
# -ngl N    : couches offloadées en GPU
# -c N      : taille du contexte
# -b N      : batch size
# -t N      : threads CPU
# --temp    : température
# --top-p   : sampling
# --grammar : contraint la sortie (grammaire GBNF)
```

5.2.5 Intégration IDE

L'intégration se fait via **`llama-server`** : tout client OpenAI-compatible (Aider, Continue, aichat, llm…) peut pointer vers `http://localhost:8080`.

5.2.6 Performance

- AVX2 / AVX-512 : tout CPU moderne supporté.
- GPU NVIDIA : CUDA recommandé (vitesse maximale).
- GPU AMD : ROCm (Linux) ou Vulkan.
- Apple Silicon : Metal (très performant).

5.3 4.3 LM Studio CLI

Nom du binaire : `lms` **Auteur** : Element Labs (LM Studio, USA) **Pays d'origine** : USA **Première sortie** : 2023 (GUI), CLI `lms` en 2024 **Licence** : propriétaire (gratuit, certains composants OSS) **Site officiel** : https://lmstudio.ai **Dépôt GitHub (CLI)** : https://github.com/lmstudio-ai/lms

5.3.1 Description

LM Studio est principalement une **application desktop** très soignée pour exécuter des modèles localement, mais l'équipe a publié `lms`, une CLI complète qui complète l'application. Elle permet de gérer les modèles, démarrer/arrêter le serveur API local, et inférer des modèles depuis le terminal.

5.3.2 Caractéristiques

- Backend : llama.cpp + MLX (sur Apple Silicon)
- Serveur OpenAI-compatible (`http://localhost:1234/v1`)
- Catalogue de modèles GGUF curés
- Gestion fine de la VRAM
- CLI scriptable, intégrable dans CI

5.3.3 Installation

Debian/Ubuntu / Fedora : la CLI est livrée avec l'application LM Studio.

```
# Télécharger AppImage
wget https://installers.lmstudio.ai/linux/x64/LM-Studio-x.y.z-Linux.AppImage
chmod +x LM-Studio-*.AppImage
./LM-Studio-*.AppImage

# Activer la CLI lms
~/.cache/lm-studio/bin/lms bootstrap
```

5.3.4 Refcard

```
lms ls                              # modèles présents
lms ls --downloaded                 # modèles téléchargés
lms ps                              # modèles chargés
lms load TheBloke/Llama-2-7B-Chat-GGUF
lms load --gpu 0.7 model            # 70 % offload
lms unload <model>
lms server start
lms server stop
lms server status
lms log stream
lms get model-id                    # téléchargement
lms version
```

5.3.5 Intégration IDE

Via API OpenAI-compatible : Continue, Cline, Aider, etc.

5.4 4.4 llamafile (Mozilla Builders)

Nom du binaire : un fichier `.llamafile` exécutable + `llamafile` **Auteur** : Justine Tunney + Mozilla Builders (USA) **Pays d'origine** : USA **Première sortie** : novembre 2023 **Licence** : Apache 2.0 (avec composants llama.cpp MIT) **Site officiel** : https://llamafile.ai **Dépôt GitHub** : https://github.com/Mozilla-Ocho/llamafile

5.4.1 Description

`llamafile` distribue un **modèle LLM + serveur dans un seul exécutable portable** (Linux, macOS, Windows, BSD, FreeBSD…), grâce à **Cosmopolitan Libc**. Le mode CLI permet d'inférer un modèle local depuis le shell sans installation de runtime, sans Python, sans Docker. Idéal pour distribuer des modèles sur USB, en démos, ou sur edge.

5.4.2 Caractéristiques

- Binaire unique 100 % offline et portable
- Mode CLI (`--cli`) + mode serveur HTTP (compatible OpenAI)
- Pipes shell
- GPU offloading (CUDA, Metal, Vulkan)
- Format GGUF natif

5.4.3 Installation

```
# Modèle pré-empaqueté (téléchargé d'un coup)
wget https://huggingface.co/Mozilla/Llama-3.2-3B-Instruct-llamafile/resolve/main/Llama-3.2-3B-Instruct.Q6_
chmod +x Llama-3.2-3B-Instruct.Q6_K.llamafile
./Llama-3.2-3B-Instruct.Q6_K.llamafile --cli -p "Bonjour"

# Outil llamafile seul (pour wrapper d'autres GGUF)
curl -L -o llamafile https://github.com/Mozilla-Ocho/llamafile/releases/latest/download/llamafile
chmod +x llamafile && sudo mv llamafile /usr/local/bin/
```

5.4.4 Refcard

```
./modele.llamafile                          # serveur web (OpenAI-compat)
./modele.llamafile --cli -p "Explique RAID"
./modele.llamafile --cli -f prompt.txt
./modele.llamafile --server --host 0.0.0.0 --port 8080
./modele.llamafile -ngl 35                  # offload GPU
./modele.llamafile --temp 0.7 --top-p 0.9
./modele.llamafile -c 8192                  # context window
echo "log..." | ./modele.llamafile --cli -p "explique"
```

5.4.5 Cas d'usage

- LLM offline portable sur USB
- Démos sans Internet
- Déploiement edge (Raspberry Pi 5, Jetson…)
- Backend pour `llm`, `aichat`, `mods` via endpoint OpenAI

5.5 4.5 vLLM

Nom du binaire : `vllm` **Auteur** : équipe vLLM (Sky Lab, UC Berkeley) **Pays d'origine** : USA **Première sortie** : juin 2023 **Licence** : Apache 2.0 **Site officiel** : https://docs.vllm.ai **Dépôt GitHub** : https://github.com/vllm-project/vllm

5.5.1 Description

vLLM est un **moteur d'inférence haute-performance** pour serveurs (GPU NVIDIA/AMD/Intel). Il introduit le **PagedAttention** (gestion de la mémoire KV-cache façon mémoire virtuelle d'OS) et le **continuous batching**, ce qui en fait l'un des serveurs les plus rapides en production pour des modèles non quantifiés. Plutôt orienté serveur que poste de travail.

5.5.2 Caractéristiques

- PagedAttention (mémoire KV optimisée)
- Continuous batching
- API OpenAI-compatible
- Quantization (AWQ, GPTQ, FP8, INT4)
- Multi-LoRA, prefix caching
- Distributed inference (multi-GPU, multi-node)
- Support Hugging Face natif

5.5.3 Installation

```
# Debian/Ubuntu / Fedora (CUDA recommandé)
pip install vllm
# ou
pipx install vllm
# ou Docker
docker run --runtime nvidia --gpus all \
  -v ~/.cache/huggingface:/root/.cache/huggingface \
  -p 8000:8000 --ipc=host \
  vllm/vllm-openai:latest \
  --model meta-llama/Meta-Llama-3.1-8B-Instruct
```

5.5.4 Refcard

```
# Serveur API
vllm serve meta-llama/Meta-Llama-3.1-8B-Instruct \
  --port 8000 --host 0.0.0.0 --tensor-parallel-size 1

# CLI inférence
python -m vllm.entrypoints.cli.main chat \
  --model Qwen/Qwen2.5-7B-Instruct \
  --message "Bonjour"

# Benchmark
python benchmarks/benchmark_throughput.py --model ...
```

5.5.5 Intégration IDE

API OpenAI-compatible : utilisable depuis Aider, Continue, aichat.

5.5.6 Cas d'usage

Serveur d'inférence en production, multi-utilisateurs, latence < 100 ms.

5.6 4.6 GPT4All

Nom du binaire : `gpt4all-cli` (et application desktop) **Auteur** : Nomic AI (USA) **Pays d'origine** : USA **Première sortie** : mars 2023 **Licence** : MIT **Site officiel** : https://gpt4all.io **Dépôt GitHub** : https://github.com/nomic-ai/gpt4all

5.6.1 Description

GPT4All est l'un des projets historiques d'inférence locale grand public. Initialement focalisé sur le CPU (modèles 7B Q4 sur ordinateurs ordinaires), il a ajouté le support GPU (Vulkan), une bibliothèque de modèles curée, et un mode RAG (LocalDocs) qui permet de discuter avec des documents locaux. La distribution principale est une **GUI desktop**, mais des bindings CLI et Python existent.

5.6.2 Caractéristiques

- CPU & GPU (Vulkan)
- LocalDocs RAG (PDF, Markdown, code)
- Bibliothèque de modèles
- Bindings Python, Node.js, Go, .NET

5.6.3 Installation

```
# Debian/Ubuntu / Fedora
pip install gpt4all
# Application desktop
wget https://gpt4all.io/installers/gpt4all-installer-linux.run
chmod +x gpt4all-installer-linux.run
./gpt4all-installer-linux.run
```

5.6.4 Refcard

```
# CLI Python (pas de binaire CLI standalone à proprement parler)
python - <<'EOF'
from gpt4all import GPT4All
model = GPT4All("Meta-Llama-3-8B-Instruct.Q4_0.gguf")
with model.chat_session():
    print(model.generate("Bonjour, qui es-tu ?"))
EOF
```

5.6.5 Cas d'usage

Postes de travail Windows/Linux/macOS sans GPU, démos, RAG local sur fichiers personnels.

5.7 4.7 LocalAI

Nom du binaire : `local-ai` **Auteur** : Ettore Di Giacinto (mudler), Italie **Pays d'origine** : Italie **Première sortie** : avril 2023 **Licence** : MIT **Site officiel** : https://localai.io **Dépôt GitHub** : https://github.com/mudler/LocalAI

5.7.1 Description

LocalAI est un **drop-in replacement de l'API OpenAI** : il expose les endpoints `/v1/chat/completions`, `/v1/embeddings`, `/v1/audio/transcriptions`, `/v1/images/generations` etc., en utilisant des modèles locaux derrière. Il supporte LLM, génération d'images (Stable Diffusion), TTS/STT (Whisper, Bark, Piper), embeddings — c'est une **plateforme tout-en-un** pour qui veut héberger toute l'IA générative chez soi.

5.7.2 Caractéristiques

- API OpenAI complète (chat, embed, image, audio, vision)
- Backends : llama.cpp, vLLM, transformers, Stable Diffusion, Whisper
- Galerie de modèles (`local-ai models install ...`)
- Federated mode (cluster)
- WebUI intégrée
- Support GGUF, ONNX, etc.

5.7.3 Installation

```
# Binaire
curl https://localai.io/install.sh | sh

# Docker
docker run -ti -p 8080:8080 --gpus all \
  -v $PWD/models:/build/models \
  localai/localai:latest-aio-gpu-nvidia-cuda-12

# Helm (Kubernetes)
helm repo add go-skynet https://go-skynet.github.io/helm-charts/
helm install local-ai go-skynet/local-ai
```

5.7.4 Refcard

```
local-ai models install llama-3.1-8b-instruct
local-ai run                    # démarre serveur (port 8080)
local-ai models list
local-ai models delete <id>
local-ai api                    # endpoints
```

5.7.5 Intégration IDE

API OpenAI-compatible : compatible avec **tout** client OpenAI. Particulièrement adapté aux organisations souhaitant un hub IA local.

5.8 4.8 Jan

Nom du binaire : `jan` (CLI), application desktop **Auteur** : Jan / Menlo Research (Vietnam, équipe distribuée) **Pays d'origine** : Vietnam **Première sortie** : 2024 **Licence** : AGPL v3 **Site officiel** : https://jan.ai **Dépôt GitHub** : https://github.com/janhq/jan

5.8.1 Description

Jan se présente comme une **alternative open-source à ChatGPT 100 % offline**. Application desktop élégante (Tauri) avec un bibliothèque de modèles, mais aussi un serveur Cortex (en CLI) qui peut tourner seul. Contrairement à LM Studio, Jan est entièrement open-source.

5.8.2 Caractéristiques

- Backend Cortex (llama.cpp + TensorRT-LLM)
- Bibliothèque de modèles
- Threads, assistants
- API OpenAI-compatible

5.8.3 Installation

```
# Debian/Ubuntu (.deb)
wget https://github.com/janhq/jan/releases/latest/download/jan-linux-amd64.deb
sudo apt install -y ./jan-linux-amd64.deb

# Fedora (.rpm)
sudo rpm -i https://github.com/janhq/jan/releases/latest/download/jan-linux-x86_64.rpm

# AppImage
wget https://github.com/janhq/jan/releases/latest/download/jan-linux-x86_64.AppImage
chmod +x jan-linux-x86_64.AppImage && ./jan-linux-x86_64.AppImage
```

5.8.4 Refcard CLI Cortex

```
cortex models pull llama3.2:3b-gguf-q4-km
cortex models list
cortex run llama3.2:3b
cortex serve --port 1337
cortex ps
cortex stop llama3.2:3b
```

5.9 4.9 RamaLama

Nom du binaire : `ramalama` **Auteur** : Containers / Red Hat (USA) **Pays d'origine** : USA **Première sortie** : 2024 **Licence** : MIT **Dépôt GitHub** : https://github.com/containers/ramalama

5.9.1 Description

RamaLama est l'outil sponsorisé par **Red Hat** pour exécuter des modèles LLM **dans des conteneurs OCI**. La philosophie : *« faire pour les modèles ce que Podman fait pour les containers »*. Il télécharge l'image OCI appropriée (CUDA, ROCm, CPU) en fonction du matériel détecté, puis exécute le modèle dedans.

5.9.2 Caractéristiques

- Backend conteneurisé (Podman/Docker)
- Détection automatique GPU (NVIDIA/AMD/Intel)
- Stockage modèles en OCI Artifacts (compatible registres)
- Compatible llama.cpp et vLLM
- Sécurité par conteneur (rootless)

5.9.3 Installation

Fedora :

```
sudo dnf install -y ramalama
```

Debian/Ubuntu :

```
pipx install ramalama
# ou
sudo apt install -y python3-pip podman
pip install --user ramalama
```

5.9.4 Refcard

```
ramalama pull llama3.3
ramalama list
ramalama run llama3.3
ramalama serve llama3.3 --port 8080
ramalama stop llama3.3
ramalama bench llama3.3
ramalama info                        # matériel détecté
ramalama containers
ramalama push my-model oci://registry.example.com/llm/my-model
```

5.9.5 Cas d'usage

Postes Linux Red Hat / Fedora, environnements DevOps containerisés, déploiement Kubernetes (`ramalama --podman --k8s`).

5.10 4.10 KoboldCpp

Nom du binaire : `koboldcpp` **Auteur** : LostRuins (communauté open-source) **Pays d'origine** : international **Première sortie** : 2023 **Licence** : AGPL v3 **Dépôt GitHub** : https://github.com/LostRuins/koboldcpp

5.10.1 Description

KoboldCpp est une **distribution alternative de llama.cpp** orientée *roleplay* / fiction interactive (héritage de KoboldAI), mais désormais utilisable comme runner LLM générique. Il fournit un binaire unique (Python embarqué via PyInstaller), avec UI web et API OpenAI / KoboldAI / Ollama, plus support Stable Diffusion et Whisper intégrés.

5.10.2 Caractéristiques

- Binaire unique (~80 Mo)
- llama.cpp + Stable Diffusion + Whisper + TTS
- API multi-formats (OpenAI, KoboldAI, Ollama)
- Mode CLI ou mode UI web

5.10.3 Installation

```
# Debian/Ubuntu / Fedora (binaire)
wget https://github.com/LostRuins/koboldcpp/releases/latest/download/koboldcpp-linux-x64-cuda1210
chmod +x koboldcpp-linux-x64-cuda1210
./koboldcpp-linux-x64-cuda1210 modele.gguf
```

5.10.4 Refcard

```
./koboldcpp model.gguf --port 5001 --gpulayers 99
./koboldcpp --help
./koboldcpp --usecublas --skiplauncher --remotetunnel
./koboldcpp --whispermodel ggml-medium.bin --sdmodel sd15.safetensors
```

5.11 4.11 text-generation-webui (Oobabooga)

Nom du binaire : `start_linux.sh`, lancement Python **Auteur** : Oobabooga (USA) **Première sortie** : décembre 2022 **Licence** : AGPL v3 **Dépôt GitHub** : https://github.com/oobabooga/text-generation-webui

5.11.1 Description

Oobabooga est l'**équivalent Stable Diffusion WebUI pour les LLM** : très étendue, très configurable, supporte tous les backends (llama.cpp, ExLlamaV2, Transformers, AutoGPTQ, MLX). Plus orientée laboratoire / expérimentation que production.

5.11.2 Installation

```
git clone https://github.com/oobabooga/text-generation-webui
cd text-generation-webui
./start_linux.sh   # script tout-en-un
# Mode API : --api --listen
```

5.11.3 Refcard

```
./start_linux.sh --api --listen
python download-model.py meta-llama/Llama-3.1-8B-Instruct
python server.py --model llama-3.1-8B-Instruct --api
```

5.12 4.12 mlx-lm (Apple)

Nom du binaire : `mlx_lm.generate`, `mlx_lm.server` **Auteur** : Apple Machine Learning Research **Pays d'origine** : USA **Première sortie** : 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/ml-explore/mlx-examples

5.12.1 Description

MLX est le **framework d'apprentissage automatique d'Apple optimisé pour Apple Silicon** (M1/M2/M3/M4). `mlx-lm` est le module dédié à l'inférence/fine-tuning des LLM. Bien que principalement destiné à macOS, il est mentionné ici pour exhaustivité — sur Linux x86, préférer llama.cpp ou vLLM.

5.12.2 Installation

```
pip install mlx-lm  # uniquement Apple Silicon
```

5.12.3 Refcard

```
mlx_lm.generate --model mlx-community/Meta-Llama-3.1-8B-Instruct-4bit \
  --prompt "Bonjour" --max-tokens 256
mlx_lm.server --model ... --port 8080
mlx_lm.lora --model ... --train --data ...
```

5.13 4.13 Outils audio / image en CLI (bonus IA générative)

5.13.1 4.13.1 whisper.cpp

Nom du binaire : `whisper-cli`, `whisper-server` **Auteur** : Georgi Gerganov + communauté **Première sortie** : septembre 2022 **Licence** : MIT **Dépôt** : https://github.com/ggml-org/whisper.cpp

Implémentation C/C++ pure de **Whisper** (modèle de reconnaissance vocale d'OpenAI). Permet la **transcription audio en CLI**.

```
# Compilation
git clone https://github.com/ggml-org/whisper.cpp
cd whisper.cpp && cmake -B build && cmake --build build -j

# Téléchargement modèle
bash ./models/download-ggml-model.sh medium

# Transcription
./build/bin/whisper-cli -m models/ggml-medium.bin -f audio.wav -l fr
./build/bin/whisper-cli -m ... -f audio.wav --output-srt --output-vtt
```

5.13.2 4.13.2 piper

Nom du binaire : `piper` **Auteur** : Michael Hansen (Rhasspy, USA) **Licence** : MIT **Dépôt** : https://github.com/rhasspy/piper

Synthèse vocale (TTS) **rapide et de bonne qualité** en CLI, multi-langues (français inclus).

```
# Installation
wget https://github.com/rhasspy/piper/releases/latest/download/piper_linux_x86_64.tar.gz
tar -xzf piper_linux_x86_64.tar.gz

# Téléchargement voix française
wget https://huggingface.co/rhasspy/piper-voices/resolve/main/fr/fr_FR/upmc/medium/fr_FR-upmc-medium.onnx
wget https://huggingface.co/rhasspy/piper-voices/resolve/main/fr/fr_FR/upmc/medium/fr_FR-upmc-medium.onnx.

# Synthèse
echo "Bonjour, je suis Piper." | ./piper/piper \
  --model fr_FR-upmc-medium.onnx --output_file out.wav
```

5.13.3 4.13.3 stable-diffusion.cpp

Nom du binaire : `sd` **Auteur** : leejet + communauté **Licence** : MIT **Dépôt** : https://github.com/leejet/stable-diffusion.cpp

Implémentation C/C++ de Stable Diffusion. Permet la génération d'images en CLI sans Python.

```
git clone --recursive https://github.com/leejet/stable-diffusion.cpp
cd stable-diffusion.cpp && cmake -B build -DSD_CUDA=ON && cmake --build build -j
./build/bin/sd -m model.safetensors -p "un chat astronaute" -o out.png \
  --steps 30 --cfg-scale 7 -W 512 -H 512
```

5.13.4 4.13.4 Coqui TTS

Nom du binaire : `tts` **Auteur** : Coqui.ai (Allemagne) **Licence** : MPL-2.0 **Dépôt** : https://github.com/idiap/coqui-ai-TTS (le projet officiel a été repris par Idiap)

Synthèse vocale neuronale, multi-modèles (XTTSv2 multilingue, VITS, Tacotron2…).

```
pipx install TTS
tts --list_models
```

```
tts --text "Bonjour" --model_name tts_models/fr/mai/tacotron2-DDC --out_path out.wav
tts-server --model_name tts_models/fr/mai/tacotron2-DDC
```

5.14 4.14 Synthèse — Runners locaux

Outil	Backend	API OpenAI	GPU	Conteneur natif	Multimodal	OS
Ollama	llama.cpp	Oui	NVIDIA/AMD/Apple	Docker	Oui vision	Lin
llama.cpp	natif	Oui (server)	tous	Non	Oui vision	un
LM Studio (lms)	llama.cpp + MLX	Oui	tous	Non	Oui	Lin
llamafile	llama.cpp	Oui (server)	NVIDIA/Apple	Non	Oui LLaVA	un
vLLM	natif	Oui	NVIDIA/AMD/Intel	Docker	Oui	Lin
GPT4All	llama.cpp	Oui partiel	Vulkan	Non	Non	Lin
LocalAI	multi	Oui complète	tous	Docker	Oui + image/audio	Lin
Jan / Cortex	llama.cpp	Oui	tous	Non	Non	Lin
RamaLama	llama.cpp/vLLM	Oui	tous	**Podman natif**	Non	Lin
KoboldCpp	llama.cpp+SD	multi	NVIDIA/AMD	Non	Oui + image	un
Oobabooga	multi	Oui	NVIDIA	Non	Oui	Lin
mlx-lm	MLX	Oui	Apple Silicon only	Non	Non	ma

Chapter 6

Partie V — Outils CLI spécialisés

Cette partie regroupe les outils qui ne sont ni des agents codeurs complets, ni des chats LLM génériques, mais des **outils spécialisés** sur une tâche précise : génération de messages de commit, conversion langage naturel → commande shell, évaluation de prompts, etc.

6.1 5.1 aicommits

Nom du binaire : `aicommits` **Auteur** : Hassan El Mghari (Nutlope), USA **Première sortie** : février 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/Nutlope/aicommits

6.1.1 Description

aicommits génère automatiquement des **messages de commit Git** en analysant le diff *staged* via l'API OpenAI. Il s'intègre dans le workflow Git classique en proposant un message validable avant le commit, avec support de plusieurs styles (conventional commits) et langues.

6.1.2 Caractéristiques

- Génération basée sur `git diff --cached`
- Support **Conventional Commits** (`--type conventional`)
- Multi-langues (anglais, français, japonais, etc.)
- Plusieurs commits proposés (`--generate N`)
- Hook Git natif (`aicommits hook install`)

6.1.3 Installation

```
# Debian/Ubuntu / Fedora (npm)
sudo npm install -g aicommits
# ou pnpm
pnpm add -g aicommits
```

6.1.4 Refcard

```
aicommits config set OPENAI_KEY=sk-...
aicommits config set locale=fr
aicommits config set type=conventional
aicommits                    # génère un message
aicommits --generate 3       # 3 propositions
aicommits --all              # stage tous les changements
```

```
aicommits --exclude '*.lock' # exclut des fichiers
aicommits hook install       # hook prepare-commit-msg
```

6.2 5.2 opencommit (oco)

Nom du binaire : `oco` **Auteur** : Dima Sukharev (di-sukharev), Estonie **Première sortie** : mars 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/di-sukharev/opencommit

6.2.1 Description

OpenCommit est un générateur de messages de commit Node.js **multi-fournisseurs** (OpenAI, Anthropic, Azure, Ollama, Gemini, Flowise). Il supporte les emojis Gitmoji, les Conventional Commits et le contexte multi-fichiers, avec un mode *auto-fix* pour le linting des messages.

6.2.2 Installation

```
sudo npm install -g opencommit
```

6.2.3 Refcard

```
oco config set OCO_API_KEY=sk-...
oco config set OCO_AI_PROVIDER=openai
oco config set OCO_MODEL=gpt-4o-mini
oco config set OCO_EMOJI=true
oco config set OCO_DESCRIPTION=true
oco config set OCO_LANGUAGE=fr
oco                          # commit interactif
oco --yes                    # auto-confirme
oco --fgm                    # full-gitmoji
oco hook set                 # installe le hook git
oco commitlint               # mode commitlint
```

6.3 5.3 gptcommit

Nom du binaire : `gptcommit` **Auteur** : Roger Zurawicki (zurawiki), USA **Première sortie** : janvier 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/zurawiki/gptcommit

6.3.1 Description

gptcommit est un **hook Git écrit en Rust** qui génère des messages de commit via OpenAI. Distribué comme binaire natif rapide, il s'installe directement comme `prepare-commit-msg`, sans dépendance Node ni Python, et utilise une approche **map-reduce** pour résumer les diffs volumineux.

6.3.2 Installation

```
# Debian/Ubuntu / Fedora
cargo install --locked gptcommit

# Brew
brew install zurawiki/brews/gptcommit
```

6.3.3 Refcard

```
gptcommit install              # installe hook dans le repo
gptcommit config set openai.api-key sk-...
gptcommit config set openai.model gpt-4o-mini
gptcommit config keys          # liste les clés
gptcommit prepare-commit-msg --commit-msg-file .git/COMMIT_EDITMSG
gptcommit uninstall
```

6.4 5.4 AI Shell (ai-shell)

Nom du binaire : `ai` **Auteur** : Builder.io (USA) **Première sortie** : mars 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/BuilderIO/ai-shell

6.4.1 Description

AI Shell convertit des requêtes en langage naturel en **commandes shell exécutables**. Il propose, explique et exécute la commande, avec un mode chat conversationnel.

6.4.2 Installation

```
sudo npm install -g @builder.io/ai-shell
```

6.4.3 Refcard

```
ai config set OPENAI_KEY=sk-...
ai config set MODEL=gpt-4o-mini
ai config set LANGUAGE=fr
ai "list all docker containers using more than 1GB"
ai chat                         # mode conversation
ai update                       # met à jour
ai config ui                    # config interactive
```

6.5 5.5 Butterfish

Nom du binaire : `butterfish` **Auteur** : Peter Bakkum (bakks), USA **Première sortie** : avril 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/bakks/butterfish

6.5.1 Description

Butterfish est un **wrapper de shell** écrit en Go qui injecte un assistant IA dans une session bash/zsh transparente. Tout ce que vous tapez peut être suivi de `#` pour interroger un LLM, et l'historique de la session sert de contexte. Inclut aussi des commandes utilitaires (résumé, prompt, embedding, *goal mode*).

6.5.2 Installation

```
# Go install
go install github.com/bakks/butterfish/cmd/butterfish@latest

# Brew
brew install bakks/bakks/butterfish
```

6.5.3 Refcard

```
butterfish shell            # lance shell wrappé
# dans la session :
ls -la # explique cette sortie       (le # est le prompt IA)
butterfish prompt "..."
butterfish summarize file.log
butterfish goal "déploie l'app sur staging"
butterfish index ./src      # crée embeddings
butterfish vectorsearch "auth flow"
```

6.6 5.6 shell-ai

Nom du binaire : `q` (oui, conflit possible avec Amazon Q) **Auteur** : Rick Lamers (ricklamers / Engineer-AI Lab), Pays-Bas **Première sortie** : 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/ricklamers/shell-ai

6.6.1 Description

shell-ai est un convertisseur **langage naturel → shell** minimaliste écrit en Python qui supporte OpenAI, Azure, Ollama et HuggingFace. Très léger, configurable via variables d'environnement.

6.6.2 Installation

```
pipx install shell-ai
```

6.6.3 Refcard

```
export OPENAI_API_KEY=sk-...
export OPENAI_MODEL=gpt-4o-mini
q "find all png larger than 1MB"
q --num 3 "kill process on port 8080"

# Avec Ollama local
export OPENAI_API_BASE=http://localhost:11434/v1
export OPENAI_MODEL=llama3
q "list git branches merged into main"
```

6.7 5.7 Pieces CLI

Nom du binaire : `pieces` **Éditeur** : Pieces for Developers / Mesh Intelligent Technologies (USA) **Première sortie** : 2023 **Licence** : MIT (CLI), PiecesOS propriétaire **Site officiel** : https://pieces.app **Dépôt GitHub** : https://github.com/pieces-app/cli-agent

6.7.1 Description

Pieces CLI est l'interface en ligne de commande de la **plateforme Pieces**, qui gère un référentiel local de snippets de code enrichis par IA. Elle interagit avec **PiecesOS** (service local) pour sauvegarder, rechercher et discuter (Copilot Chat) avec ses snippets, avec support local-first (LLM Ollama).

6.7.2 Installation

```
# pipx
pipx install pieces-cli
# Brew
brew install pieces-cli

# PiecesOS Linux (.deb)
wget https://docs.pieces.app/products/meet-pieces/fundamentals/linux
# (suivre la page officielle pour le paquet)
```

6.7.3 Refcard

```
pieces onboarding
pieces login
pieces list                    # liste snippets
pieces save                    # sauve depuis clipboard
pieces ask "comment parser JSON en Go ?"
pieces chat                    # session conversationnelle
pieces search "regex"
pieces config --model llama-3
pieces run                     # exécute un snippet
```

6.8 5.8 promptfoo

Nom du binaire : `promptfoo` **Éditeur** : promptfoo, Inc. (USA) **Première sortie** : avril 2023 **Licence** : MIT **Site officiel** : https://www.promptfoo.dev **Dépôt GitHub** : https://github.com/promptfoo/promptfoo

6.8.1 Description

promptfoo est un framework d'**évaluation et de red-teaming pour prompts LLM**, distribué comme CLI Node.js. Il permet de définir des cas de test (assertions JSON-schema, similarité, factualité, modèle-as-juge), de comparer plusieurs modèles/prompts en parallèle et d'exécuter des suites de sécurité (jailbreak, PII, prompt-injection).

6.8.2 Caractéristiques

- Configuration YAML déclarative (`promptfooconfig.yaml`)
- Comparaison côte-à-côte multi-prompts/modèles
- Assertions : `contains`, `regex`, `factuality`, `llm-rubric`, `similar`, `javascript`
- Red-team : OWASP LLM Top 10, jailbreaks, harmful content
- Web UI locale (`promptfoo view`)
- Intégration CI (GitHub Actions)
- Providers : OpenAI, Anthropic, Vertex, Bedrock, Ollama, HuggingFace, custom

6.8.3 Installation

```
# npm
sudo npm install -g promptfoo
# ou one-shot
npx promptfoo@latest init
# Brew
brew install promptfoo
```

6.8.4 Refcard

```
promptfoo init                  # crée promptfooconfig.yaml
promptfoo eval                  # lance l'évaluation
promptfoo eval -c config.yaml -j 8 -o results.html
promptfoo view                  # UI web sur localhost
promptfoo redteam init          # init suite red-team
promptfoo redteam run
promptfoo share                 # partage cloud
promptfoo cache clear
promptfoo generate dataset      # synthétique
```

6.8.5 Intégration IDE

Extension VSCode officielle « promptfoo ».

6.8.6 Cas d'usage

QA prompts, A/B testing modèles, audits sécurité LLM, intégration CI bloquant les régressions de qualité.

6.9 5.9 ata (Ask The Terminal Anything)

Nom du binaire : `ata` **Auteur** : Rik Huijzer (rikhuijzer), Pays-Bas **Licence** : MIT **Dépôt GitHub** : https://github.com/rikhuijzer/ata

6.9.1 Description

`ata` est un CLI Julia/Rust minimaliste qui transforme la sortie de votre terminal en **contexte pour poser des questions à un LLM**. Idéal pour debugger des erreurs sans copier-coller manuellement.

6.9.2 Installation

```
# Téléchargement binaire depuis releases GitHub
wget https://github.com/rikhuijzer/ata/releases/latest/download/ata-linux-x64
chmod +x ata-linux-x64 && sudo mv ata-linux-x64 /usr/local/bin/ata
```

6.9.3 Refcard

```
ata "explain this error"
ata --model gpt-4o
ata config
```

6.10 5.10 hai (Braintrust / variantes)

Nom du binaire : `hai` **Éditeur** : Braintrust Data (USA) — variante principale **Licence** : MIT **Dépôt GitHub probable** : https://github.com/braintrustdata/hai

6.10.1 Description

hai est un agent CLI **REPL multi-modèles** marketé par Braintrust comme un *« REPL multi-modèle »*. Il permet d'enchaîner appels modèles, exécution shell et lecture de fichiers dans une session terminale, avec support de multiples providers (OpenAI, Anthropic, Google, Braintrust Proxy).

6.10.2 Installation

```
# Cargo
cargo install hai-cli
# ou
curl -fsSL https://hai.sh/install.sh | sh
```

6.10.3 Refcard

```
hai                          # session REPL
/model claude-sonnet-4.5
/exec ls -la
/file lire ./src/main.rs
/save conversation.md
hai run "résume ce repo"
hai --provider braintrust
```

> **Avertissement** : plusieurs projets nommés *hai* coexistent (notamment des projets japonais et asiatiques). Vérifier l'éditeur exact à l'installation.

6.11 5.11 Khoj

Nom du binaire : `khoj` **Auteur** : Debanjum Singh Solanky et Saba (Khoj AI, USA/Inde) **Licence** : AGPL v3 (open core) + offre cloud **Dépôt GitHub** : https://github.com/khoj-ai/khoj

6.11.1 Description

Khoj est un **assistant personnel** open-source qui indexe vos notes (Markdown, Org-mode), PDF, GitHub, e-mails Notion… puis répond à vos questions en RAG. Il propose une CLI, des plugins (Obsidian, Emacs, Logseq), et une UI web. Très utilisé par les *knowledge workers* qui veulent un *« second cerveau IA »*.

6.11.2 Installation

```
pipx install khoj
khoj --anonymous-mode  # premier lancement
```

6.11.3 Refcard

```
khoj                                # démarre serveur (port 42110)
khoj --host 0.0.0.0
khoj --no-gui                       # serveur sans GUI
khoj --version
# Indexation
khoj index --content-type markdown --files ~/notes
```

6.11.4 Intégrations

- **Obsidian** : plugin officiel
- **Emacs** : `khoj.el`
- **Logseq** : plugin
- **Web UI** : dashboard intégré
- **Desktop app** : Tauri

6.12 5.12 Tabby

Nom du binaire : `tabby` (serveur), plugins IDE **Auteur** : TabbyML (USA) **Licence** : Apache 2.0 **Site officiel** : https://tabby.tabbyml.com **Dépôt GitHub** : https://github.com/TabbyML/tabby

6.12.1 Description

Tabby est l'**alternative self-hosted à Copilot** : un serveur d'inférence local pour code completion et chat, optimisé GPU, qui s'utilise via plugins IDE (VSCode, JetBrains, Vim, Emacs). Cible : organisations qui veulent un Copilot privé.

6.12.2 Installation

Debian/Ubuntu / Fedora (Docker recommandé) :

```
docker run -it --gpus all -p 8080:8080 \
  -v $HOME/.tabby:/data \
  registry.tabbyml.com/tabbyml/tabby \
  serve --model StarCoder-1B --device cuda
```

6.12.3 Refcard

```
tabby serve --model StarCoder-1B --device cuda
tabby download --model TabbyML/StarCoder-1B
tabby info
```

6.12.4 Intégration IDE

- **VSCode** : extension Tabby officielle
- **JetBrains** : plugin officiel
- **Vim/Neovim** : `tabby.vim`
- **Emacs** : `tabby.el`

6.13 5.13 Synthèse — Outils spécialisés

Outil	Rôle	Langage	Multi-providers
aicommits	Messages de commit	Node	OpenAI
opencommit (oco)	Messages de commit	Node	OpenAI, Anthropic, Ollama…
gptcommit	Messages de commit (Rust)	Rust	OpenAI
AI Shell	NL → shell	Node	OpenAI
Butterfish	Shell wrapper	Go	OpenAI
shell-ai (q)	NL → shell	Python	OpenAI-compat, Ollama
Pieces CLI	Snippet manager	Python	Local (Ollama) + cloud
promptfoo	Eval / red-team	Node	tous
ata	Q&A terminal	Rust	OpenAI
hai (Braintrust)	REPL multi-modèles	Rust	OpenAI, Anthropic, Google
Khoj	RAG personnel	Python	OpenAI, Anthropic, locaux
Tabby	Code completion self-hosted	Rust	local (StarCoder…)

Chapter 7

Partie VI — Frameworks d'agents avec usage CLI

Cette partie couvre les **frameworks** pour construire des agents LLM (orchestration, mémoire, outils, multi-agents). Ils ne sont pas tous strictement *CLI-first*, mais ils proposent tous une CLI au moins pour l'initialisation, l'exécution ou le déploiement.

7.1 6.1 AutoGPT

Nom du binaire / lancement : `./run.sh` (Classic) ou `docker compose` (Platform) **Auteur** : Significant Gravitas (Toran Bruce Richards), Royaume-Uni **Pays d'origine** : Royaume-Uni **Première sortie** : mars 2023 **Licence** : MIT (Classic) / Polyform Shield 1.0.0 (Platform serveur) **Site officiel** : https://agpt.co **Dépôt GitHub** : https://github.com/Significant-Gravitas/AutoGPT

7.1.1 Description

AutoGPT est l'un des **premiers agents autonomes** « GPT-4 in a loop » (mars 2023). Depuis 2024, le projet a pivoté vers une plateforme web avec un *builder* visuel (AutoGPT Platform), mais conserve un mode CLI Classic (`autogpt-classic`) pour exécuter des agents autonomes en ligne de commande, avec mémoire vectorielle et plug-ins.

7.1.2 Caractéristiques

- Boucle planification / exécution autonome
- Mémoire long-terme (Pinecone, Redis, fichiers)
- Plugins (browser, GitHub, fichiers)
- Frontend Next.js + backend FastAPI (Platform)
- Marketplace d'agents

7.1.3 Installation

Platform (Docker recommandé) :

```
git clone https://github.com/Significant-Gravitas/AutoGPT
cd AutoGPT/autogpt_platform
docker compose up -d --build
# UI sur http://localhost:3000
```

Classic (CLI) :

```
git clone https://github.com/Significant-Gravitas/AutoGPT
cd AutoGPT/classic/original_autogpt
./run.sh
```

7.1.4 Refcard

```
./run.sh                        # mode interactif
./run.sh --continuous
./run.sh --ai-settings ai.yaml
./run.sh --skip-reprompt
# Platform
docker compose up
```

7.2 6.2 BabyAGI

Nom du binaire : `python babyagi.py` **Auteur** : Yohei Nakajima (Japon / USA) **Première sortie** : avril 2023 **Licence** : MIT **Dépôt GitHub** : https://github.com/yoheinakajima/babyagi

7.2.1 Description

BabyAGI est un agent **minimaliste** (~140 lignes initialement) démontrant la décomposition de tâches en sous-tâches via LLM + base vectorielle. Initialement script unique, le projet a évolué vers BabyAGI 2.0 (framework de *self-building autonomous agents*) avec function-calling et stockage SQLite.

7.2.2 Caractéristiques

- Boucle *task creation* → *prioritization* → *execution*
- Mémoire vectorielle (Chroma, Pinecone)
- Self-building functions (v2)
- Dashboard web léger (v2)

7.2.3 Installation

```
git clone https://github.com/yoheinakajima/babyagi
cd babyagi
pip install -r requirements.txt
cp .env.example .env  # configurer OPENAI_API_KEY
python babyagi.py
```

7.3 6.3 CrewAI

Nom du binaire : `crewai` **Auteur** : crewAI Inc. (João Moura), USA **Première sortie** : octobre 2023 **Licence** : MIT (open source) + offre Enterprise **Site officiel** : https://crewai.com **Dépôt GitHub** : https://github.com/crewAIInc/crewAI

7.3.1 Description

CrewAI est un framework Python pour orchestrer **plusieurs agents LLM collaboratifs** avec rôles, objectifs et outils définis. Il dispose d'une CLI officielle (`crewai`) pour scaffolder, exécuter et déployer des *crews* (équipes d'agents) ainsi que des *flows* (workflows déterministes).

7.3.2 Caractéristiques

- Multi-agents avec rôles / délégation
- Outils intégrés (web search, fichiers, code)
- *Flows* (workflow event-driven)
- Hierarchical / sequential process
- Intégration LangChain tools
- Mode train et test

7.3.3 Installation

```
pipx install crewai
# extras outils
pipx install 'crewai[tools]'
```

7.3.4 Refcard

```
crewai create crew my_crew         # scaffold
crewai create flow my_flow
cd my_crew
crewai install                     # installe deps
crewai run                         # exécute le crew
crewai train -n 3 -f training.json
crewai test -n 5 -m gpt-4o
crewai chat                        # mode interactif
crewai deploy create               # CrewAI Enterprise
crewai tool install <tool>
```

7.4 6.4 MetaGPT

Nom du binaire : `metagpt` **Auteur** : DeepWisdom (geekan), Chine **Première sortie** : juin 2023 **Licence** : MIT **Site officiel** : https://www.deepwisdom.ai **Dépôt GitHub** : https://github.com/geekan/MetaGPT

7.4.1 Description

MetaGPT modélise une **équipe logicielle complète** (PM, architecte, ingénieur, QA) en agents LLM coopérants. À partir d'un *one-liner* (« crée-moi une app TodoList »), il produit specs, design, code et tests. Inclut une CLI pour lancer les *company runs* et un système de SOPs (Standard Operating Procedures). Le projet est notable pour avoir popularisé l'idée de *« company of agents »*.

7.4.2 Caractéristiques

- Agents avec SOPs simulant rôles d'entreprise
- Génération full-stack (code + docs + tests)
- Data Interpreter (analyse de données autonome)
- Intégration multi-LLM
- Coût-tracking par agent

7.4.3 Installation

```
pip install --upgrade metagpt
# ou Docker
docker pull metagpt/metagpt:latest
docker run --rm metagpt/metagpt:latest "create a snake game"
metagpt --init-config  # crée ~/.metagpt/config2.yaml
```

7.4.4 Refcard

```
metagpt "create a 2048 game in pygame"
metagpt "build a CRM app" --investment 5.0 --n-round 10
metagpt --code-review True --run-tests True
# Data Interpreter
python -m metagpt.roles.di.data_interpreter
```

7.5 6.5 SuperAGI

Nom du binaire : Service Docker + dashboard web (CLI accessoire) **Éditeur** : TransformerOptimus / SuperAGI, Inde **Première sortie** : mai 2023 **Licence** : MIT **Site officiel** : https://superagi.com **Dépôt GitHub** : https://github.com/TransformerOptimus/SuperAGI

7.5.1 Description

SuperAGI est une plateforme open-source d'orchestration d'agents autonomes avec un **dashboard graphique** (Next.js), une mémoire long-terme, des outils (toolkits installables) et un système de *Concurrent Agents*. Distribuée principalement via Docker Compose, elle dispose d'une API REST pour pilotage CLI/CI.

7.5.2 Installation

```
git clone https://github.com/TransformerOptimus/SuperAGI
cd SuperAGI
cp config_template.yaml config.yaml  # éditer clés API
docker compose -f docker-compose.yaml up --build
# UI : http://localhost:3000
```

7.5.3 Refcard

```
# CLI via API REST
curl -X POST localhost:8001/api/agents \
  -H "Content-Type: application/json" \
  -d '{"name":"researcher","goal":["..."]}'
docker compose logs -f backend
```

7.6 6.6 AgentGPT

Nom du binaire : Application web (Next.js) **Éditeur** : Reworkd AI (USA) **Première sortie** : avril 2023 **Licence** : GPL-3.0 **Dépôt GitHub** : https://github.com/reworkd/AgentGPT

7.6.1 Description

AgentGPT est une web app (Next.js + tRPC + Prisma) qui permet de **configurer un agent autonome via interface graphique**. Bien qu'elle ne soit pas une CLI à proprement parler, elle se déploie via `docker compose` et expose une API. Version cloud SaaS sur agentgpt.reworkd.ai.

7.6.2 Installation

```
git clone https://github.com/reworkd/AgentGPT
cd AgentGPT
./setup.sh          # script interactif
# ou Docker
docker compose -f docker-compose.yml up
```

7.7 6.7 LangChain CLI

Nom du binaire : `langchain` **Éditeur** : LangChain Inc. (USA) **Première sortie** : 2023 **Licence** : MIT **Site officiel** : https://python.langchain.com **Dépôt GitHub** : https://github.com/langchain-ai/langchain

7.7.1 Description

La CLI LangChain est l'utilitaire de **scaffolding** officiel pour LangChain (Python) : elle crée des projets **LangServe** (templates d'apps déployables), gère les migrations entre versions majeures (v0.1 → v0.2 → v0.3) et publie sur LangChain Hub.

7.7.2 Installation

```
pipx install langchain-cli
pipx install 'langchain-cli[serve]'
```

7.7.3 Refcard

```
langchain app new my-app
cd my-app
langchain app add rag-chroma-private
langchain serve                 # lance LangServe
langchain template new my-template
langchain migrate ./src         # migration de code
langchain hub push my-prompt
```

7.8 6.8 LangGraph CLI

Nom du binaire : `langgraph` **Éditeur** : LangChain Inc. (USA) **Première sortie** : 2024 **Licence** : MIT **Site officiel** : https://langchain-ai.github.io/langgraph **Dépôt GitHub** : https://github.com/langchain-ai/langgraph

7.8.1 Description

LangGraph CLI est l'outil de développement et de déploiement pour **LangGraph** (framework de graphes d'agents stateful). Elle propose un serveur de dev local (`langgraph dev`) avec **Studio UI** intégrée, des builds Docker, et le déploiement vers LangGraph Cloud / Platform.

7.8.2 Installation

```
pipx install langgraph-cli
pipx install 'langgraph-cli[inmem]'
```

7.8.3 Refcard

```
langgraph new --template react-agent-python my-graph
cd my-graph
langgraph dev                    # serveur local + Studio
langgraph dev --port 2024 --no-browser
langgraph build -t my-graph:0.1
langgraph up                     # docker-compose
langgraph dockerfile Dockerfile
langgraph deploy
```

7.8.4 Intégration IDE

LangGraph Studio (desktop app + web).

7.9 6.9 Haystack / Hayhooks

Nom du binaire : `hayhooks` (CLI principale Haystack 2.x) **Éditeur** : deepset GmbH (Allemagne) **Pays d'origine** : Allemagne **Première sortie** : Hayhooks 2023, Haystack v2 stable 2024 **Licence** : Apache 2.0 **Site officiel** : https://haystack.deepset.ai **Dépôt GitHub** : https://github.com/deepset-ai/haystack, https://github.com/deepset-ai/hayhooks

7.9.1 Description

Haystack 2.x (le framework RAG/agents de deepset) ne dispose pas d'une CLI monolithique mais d'utilitaires : `hayhooks` (déploie des Haystack pipelines comme **APIs REST**) et le module `haystack` Python pour exécution programmatique.

7.9.2 Caractéristiques

- Pipelines déployables en REST via hayhooks
- MCP server natif (Haystack 2.x)
- Composants modulaires (retrievers, generators, rankers)
- Multi-vector store (Elasticsearch, Weaviate, Pinecone, Chroma)

7.9.3 Installation

```
pip install haystack-ai
pip install hayhooks
```

7.9.4 Refcard

```
hayhooks run                        # serveur (par défaut localhost:1416)
hayhooks pipeline deploy ./pipe.yml
hayhooks pipeline deploy-files ./folder
hayhooks pipeline list
hayhooks pipeline undeploy <name>
hayhooks status
```

7.10 6.10 Letta (ex-MemGPT)

Nom du binaire : `letta` **Éditeur** : Letta (anciennement MemGPT, Charles Packer et al., spin-off UC Berkeley) **Première sortie** : MemGPT octobre 2023, rebranding Letta 2024 **Licence** : Apache 2.0 **Site officiel** : https://letta.com **Dépôt GitHub** : https://github.com/letta-ai/letta

7.10.1 Description

Letta est un framework d'agents avec **mémoire long-terme persistante**, descendant direct du papier MemGPT (2023) qui simule un *« OS »* pour LLM avec hiérarchie de mémoire (*core memory*, *archival*, *recall*). Distribué avec un serveur (FastAPI), une CLI et une UI desktop (Letta Desktop / ADE).

7.10.2 Caractéristiques

- Mémoire stratifiée (core/archival/recall)
- Persistance SQLite/Postgres
- Outils MCP, web, code
- Multi-utilisateurs
- ADE (Agent Development Environment) GUI
- API OpenAI-compatible

7.10.3 Installation

```
pip install -U letta
# ou Docker
docker run -d --name letta \
  -p 8283:8283 \
  -v ~/.letta/.persist/pgdata:/var/lib/postgresql/data \
  -e OPENAI_API_KEY=sk-... \
  letta/letta:latest
```

7.10.4 Refcard

```
letta server                        # démarre le serveur
letta run                           # mode REPL agent
letta configure                     # config interactive
letta agent list
letta agent create --name assistant --persona helpful
letta agent message <id> "hello"
letta load directory ./docs         # ingestion archival
letta source list
```

7.11 6.11 Agno (ex-Phidata)

Nom du binaire : `ag` **Éditeur** : Agno AGI (anciennement Phidata), USA / Inde **Première sortie** : Phidata 2023, Agno 2024 **Licence** : MPL-2.0 **Site officiel** : https://agno.com **Dépôt GitHub** : https://github.com/agno-agi/agno

7.11.1 Description

Agno (anciennement Phidata) est un framework Python pour construire des agents **multi-modaux performants** (texte, image, audio, vidéo) avec mémoire, knowledge bases et reasoning. La CLI `ag` permet de scaffolder des projets, lancer un *playground* web local et déployer sur AWS.

7.11.2 Caractéristiques

- Agents multi-modaux (vision, audio)
- Knowledge bases vectorielles (LanceDB, Pgvector, Chroma)
- Workflows multi-agents
- Playground UI (Next.js local)
- Déploiement AWS (ECS, RDS)
- Performance : agents instanciables en microsecondes

7.11.3 Installation

```
pip install -U agno
pip install -U "agno[cli]"
pip install -U "agno[aws]"
```

7.11.4 Refcard

```
ag init                          # auth
ag ws create                     # nouveau workspace
ag ws up                         # lance localement (Docker)
ag ws up dev:docker
ag ws up prod:aws
ag ws down
# Playground
python playground.py             # UI sur localhost:7777
```

7.12 6.12 Pydantic AI CLI (clai)

Nom du binaire : `clai` **Éditeur** : Pydantic Services (Samuel Colvin), Royaume-Uni **Première sortie** : décembre 2024 **Licence** : MIT **Site officiel** : https://ai.pydantic.dev **Dépôt GitHub** : https://github.com/pydantic/pydantic-ai

7.12.1 Description

Pydantic AI est un framework agent Python (lancé fin 2024) qui apporte **la rigueur de Pydantic au monde des agents LLM** (validation typée des outputs, dependency injection). La CLI `clai` (incluse depuis 0.0.20+) offre un REPL conversationnel multi-modèles avec markdown rendering et l'intégration Logfire.

7.12.2 Caractéristiques

- Type-safe agents (outputs Pydantic)
- Multi-modèles (OpenAI, Anthropic, Gemini, Groq, Mistral, Ollama, Bedrock, Cohere)
- Dependency injection
- Streaming structuré
- Intégration Logfire (observabilité)

7.12.3 Installation

```
pip install pydantic-ai
pipx install 'pydantic-ai[cli]'
```

7.12.4 Refcard

```
clai                                # REPL interactif
clai --model openai:gpt-4o
clai --model anthropic:claude-sonnet-4.5
clai "résume ce CSV" < data.csv
clai --code-theme monokai
/exit  /markdown  /multiline       # commandes inline
```

7.13 6.13 Semantic Kernel

Nom du binaire : pas de CLI officielle ; utilitaires `dotnet new sk-*` + extensions communautaires **Éditeur** : Microsoft (USA) **Première sortie** : mars 2023 **Licence** : MIT **Site officiel** : https://learn.microsoft.com/semantic-kernel **Dépôt GitHub** : https://github.com/microsoft/semantic-kernel

7.13.1 Description

Semantic Kernel est le SDK orchestrateur de Microsoft pour **C#, Python et Java**, permettant d'intégrer LLM + plugins (skills) + planners dans des applications. Il n'existe pas de CLI officielle Microsoft, mais l'usage CLI se fait via `dotnet` (templates `Microsoft.SemanticKernel.Templates`) ou des scripts Python.

7.13.2 Installation

```
# .NET
dotnet new install Microsoft.SemanticKernel.Templates
dotnet new sk-console -n MyAgent

# Python
pip install semantic-kernel
```

7.13.3 Refcard

```
dotnet new sk-console -n MyApp
dotnet new sk-plugin
dotnet run
git clone https://github.com/microsoft/semantic-kernel
```

7.14 6.14 Autres frameworks notables

- **DSPy** (`dspy-ai`, Stanford) — programmation déclarative de pipelines LLM. https://github.com/stanfordnlp/dspy
- **AutoGen** (Microsoft Research) — conversation multi-agents. https://github.com/microsoft/autogen
- **OpenAI Swarm** — framework d'orchestration d'agents OpenAI (en bêta). https://github.com/openai/swarm
- **Camel-AI** — multi-agents *role-playing*. https://github.com/camel-ai/camel
- **AutoChain** — alternative simplifiée à LangChain.
- **Eidolon** (eidolon-ai) — framework agentique enterprise.
- **AGNO Cloud**, **n8n** (workflows IA) — orchestration low-code.

Chapter 8

Partie VII — Tableau comparatif synthétique

Cette partie regroupe l'ensemble des outils dans des tableaux comparatifs synthétiques pour faciliter la sélection en fonction d'un besoin précis.

8.1 7.1 Vue générale

Outil	Catégorie	Licence	Langage	Pays	Multi-LLM	Agentique	Local	MCP
Claude Code	Agent codeur propriétaire	Propriétaire	Node	USA	partiel	+++	Non	natif
Codex CLI	Agent codeur propriétaire	Apache 2.0	Rust	USA	partiel	+++	Non	Oui
Gemini CLI	Agent codeur propriétaire	Apache 2.0	Node	USA	Non	+++	Non	Oui
Copilot CLI	Agent codeur propriétaire	Propriétaire	Node	USA	Oui (3+)	++	Non	Oui
Cursor Agent	Agent codeur propriétaire	Propriétaire	Node	USA	Oui	+++	Non	Oui
Cody CLI	Agent codeur propriétaire	Apache 2.0	TypeScript	USA	Oui	++	Non	Oui
Amazon Q	Agent codeur propriétaire	Propriétaire	Rust	USA	Non (Bedrock)	++	Non	Oui

Outil	Catégorie	Licence	Langage	Pays	Multi-LLM	Agentique	Local	MCP
Warp AI	Terminal IA	Propriétaire	Rust	USA	Oui	++	Non	partiel
Aider	Agent codeur OSS	Apache 2.0	Python	USA	+++	+++	Oui	Non
Plandex	Agent codeur OSS	MIT/AGPL	Go	USA	+++	+++	Oui	Non
Goose	Agent codeur OSS	Apache 2.0	Rust	USA	+++	+++	Oui	natif
OpenHands	Agent codeur OSS	MIT	Python	USA	+++	+++	Oui	Oui
Cline	Agent codeur OSS	Apache 2.0	TypeScript	USA	+++	+++	Oui	Oui
Continue	Agent codeur OSS	Apache 2.0	TypeScript	USA	+++	++	Oui	Oui
Open Interpreter	Agent codeur OSS	AGPL v3	Python	USA	++	+++	Oui	Non
gpt-engineer	Agent codeur OSS	MIT	Python	Suède	++	++	Oui	Non
gptme	Agent CLI OSS	MIT	Python	Suède	++	++	Oui	Non
SWE-agent	Agent codeur OSS	MIT	Python	USA	++	+++	Oui	Non
Mentat	Agent codeur OSS	Apache 2.0	Python	USA	Oui	Oui	Oui	Non
Crush	Agent codeur OSS	MIT	Go	USA	++	++	Oui	Oui
Roo Code	Agent codeur OSS	Apache 2.0	TypeScript	USA	+++	+++	Oui	Oui
llm	CLI LLM générique	Apache 2.0	Python	UK/USA	Oui (plugins)	Non	Oui	partiel
sgpt	CLI LLM générique	MIT	Python	OUZ/DE	Oui (OAI-compat)	Non	Oui	Non
aichat	CLI LLM générique	MIT/Apache	Rust	Chine	+++ (20+)	Oui	Oui	partiel
mods	CLI LLM générique	MIT	Go	USA	++	Non	Oui	Non

Outil	Catégorie	Licence	Langage	Pays	Multi-LLM	Agentique	Local	MCP
tgpt	CLI LLM générique	GPL v3	Go	BD	Oui (free)	Non	Oui	Non
fabric	CLI LLM générique	MIT	Go	USA	++	patterns	Oui	Non
gptscript	CLI LLM générique	Apache 2.0	Go	USA	Oui	Oui (DSL)	Oui	partiel
smartcat	CLI LLM générique	Apache 2.0	Rust	France	++	Non	Oui	Non
shai (OVH)	CLI LLM générique	Apache 2.0	Rust	France	Oui	partiel	Oui	Non
Ollama	Runner local	MIT	Go	USA	—	—	+++	—
llama.cpp	Runner local	MIT	C++	BG	—	—	+++	—
LM Studio (lms)	Runner local	Propriétaire	TypeScript	USA	—	—	+++	—
llamafile	Runner local	Apache 2.0	C++	USA	—	—	+++	—
vLLM	Runner local	Apache 2.0	Python	USA	—	—	+++	—
LocalAI	Runner local	MIT	Go	Italie	—	—	+++	—
RamaLama	Runner local	MIT	Python	USA	—	—	+++	—
aicommits	Spécialisé commits	MIT	Node	USA	Non	Non	Non	Non
opencommit	Spécialisé commits	MIT	Node	EE	Oui	Non	Oui	Non
gptcommit	Spécialisé commits	MIT	Rust	USA	Non	Non	Non	Non
AI Shell	NL → shell	MIT	Node	USA	Non	Non	Non	Non
promptfoo	Eval/redteam	MIT	Node	USA	+++	Non	Oui	Non
Khoj	RAG personnel	AGPL v3	Python	USA/IN	++	Non	Oui	Non
Tabby	Code completion self-hosted	Apache 2.0	Rust	USA	—	—	+++	—
AutoGPT	Framework agent	MIT/PolyForm	Python	UK	Oui	+++	Oui	Non

Outil	Catégorie	Licence	Langage	Pays	Multi-LLM	Agentique	Local	MCP
CrewAI	Framework agent	MIT	Python	USA	++	+++	Oui	partiel
MetaGPT	Framework agent	MIT	Python	Chine	Oui	+++	Oui	Non
LangGraph CLI	Framework agent	MIT	Python	USA	++	+++	Oui	Oui
Letta	Framework agent	Apache 2.0	Python	USA	++	+++	Oui	Oui
Agno	Framework agent	MPL 2.0	Python	USA/IN	++	+++	Oui	Oui

8.2 7.2 Choix d'outil par cas d'usage

8.2.1 Refactoring d'un projet existant (15 K lignes)

1. **Aider** (référence OSS, pipeline Git impeccable)
2. **Plandex** (planification longue durée)
3. **Claude Code** ou **Codex CLI** (modèles de pointe)
4. **Goose** (architecte+éditeur, MCP)

8.2.2 Génération de projet from-scratch

1. **gpt-engineer** (le standard)
2. **MetaGPT** (équipe simulée)
3. **Smol Developer** (minimaliste)
4. **Lovable** (cloud, hors CLI)

8.2.3 Agent autonome long-running

1. **OpenHands** (sandbox Docker complet)
2. **AutoGPT Platform**
3. **SWE-agent** (issues GitHub)
4. **Plandex** (auto-debug)

8.2.4 Filtre LLM dans un pipeline shell

1. **mods** (sortie markdown coloré)
2. **fabric** (patterns réutilisables)
3. **smartcat** (substitution in-place)
4. **llm** (sessions historisées)

8.2.5 Conversion langage naturel → commande shell

1. **gh copilot suggest**
2. **Warp AI**
3. **AI Shell** (`ai`)
4. **Butterfish** (shell wrapper)
5. **shai** (souverain EU)
6. **gorilla-cli**

8.2.6 Pas de cloud, 100 % local

1. **Ollama** + **Aider** ou **Continue**
2. **llamafile** (portable)
3. **LocalAI** (drop-in OpenAI)
4. **Tabby** (code completion)
5. **Khoj** (RAG personnel)

8.2.7 Évaluation et red-team de prompts

1. **promptfoo** (référence)
2. **Langfuse** (observabilité)
3. **Helicone**

8.2.8 Génération de messages de commit

1. **opencommit** (le plus polyvalent)
2. **aicommits**
3. **gptcommit** (Rust, hook silencieux)

8.2.9 RAG sur documents personnels

1. **Khoj**
2. **AnythingLLM**
3. **llm + plugins embeddings**
4. **aichat –rag**

8.2.10 Mémoire long-terme persistante

1. **Letta** (le successeur de MemGPT, référence)
2. **Agno** (knowledge bases)

8.2.11 Intégration MCP

1. **Claude Code** (créateur du standard)
2. **Goose** (« MCP first »)
3. **Cline / Roo Code**
4. **Cursor Agent**
5. **Codex CLI**

8.3 7.3 Comparatif des modèles économiques

Type	Outils typiques	Coût
100 % gratuit OSS	Aider, Plandex, Goose, OpenHands, Cline, llm, sgpt, aichat, Ollama…	0 € (vous payez seulement le modèle si cloud)
Freemium SaaS	Claude Code, Codex, Gemini, Copilot, Cursor	10–200 $/mois
Pay-as-you-go	API directes (Anthropic, OpenAI, Mistral, Groq)	quelques $/M tokens
Self-hosted gratuit	LocalAI + modèles HF	coût matériel uniquement
Enterprise	Tabnine, Cody Enterprise, AWS Q	30–50 $/utilisateur/mois

8.4 7.4 Sécurité et confidentialité

Niveau de sensibilité	Recommandation
Code public, hobby	n'importe quel outil cloud (économique)
Projet pro, NDA léger	Plans Business avec *zero retention* (Claude/OpenAI/Anthropic via API), Codex en mode sandbox
Code propriétaire critique	Local : Ollama + Aider, Tabby self-hosted, Continue + vLLM
Air-gap / souverain EU	shai (OVH), LocalAI, Mistral on-prem, llama.cpp
Régulé (santé, finance, défense)	self-hosting strict, audit, chiffrement, pas de logs cloud

Chapter 9

Partie VIII — Guide d'intégration IDE

Cette partie détaille les **intégrations entre CLI IA et IDE** populaires (VSCode/VSCodium, JetBrains, Neovim, Emacs). La plupart des CLI peuvent être utilisées de manière autonome dans un terminal externe, mais l'intégration IDE permet une expérience plus fluide (sélection de code partagée, ouverture automatique des diffs, raccourcis clavier dédiés).

9.1 8.1 VSCode et VSCodium

VSCode (Microsoft) et VSCodium (build sans télémétrie) sont les IDE les plus ciblés par les éditeurs d'IA.

9.1.1 Extensions officielles

Outil CLI	Extension VSCode	Lien marketplace
Claude Code	« Claude Code for VS Code »	https://marketplace.visualstudio.com/items?itemName=anthropic.claude-code
Codex CLI	« OpenAI Codex »	Marketplace MS
Gemini CLI	« Gemini Code Assist »	Marketplace MS
GitHub Copilot CLI	« GitHub Copilot »	Marketplace MS (préinstallé sur GitHub Codespaces)
Cursor Agent	utilisation native dans Cursor (fork VSCode)	—
Cody CLI	« Cody AI »	Marketplace MS
Amazon Q	« Amazon Q »	Marketplace MS
Tabnine CLI	« Tabnine AI »	Marketplace MS
Continue	« Continue »	https://marketplace.visualstudio.com/items?itemName=Continue.continue
Cline	« Cline »	https://marketplace.visualstudio.com/items?itemName=saoudrizwan.claude-dev
Roo Code	« Roo Code »	https://marketplace.visualstudio.com/items?itemName=RooVeterinaryInc.roo-cline
Aider	« Aider VSCode » (communautaire)	OpenVSX
Tabby	« Tabby AI »	Marketplace MS
promptfoo	extension officielle	Marketplace MS

9.1.2 Installation typique

```
# Liste des extensions IA recommandées
code --install-extension Continue.continue
code --install-extension saoudrizwan.claude-dev
code --install-extension anthropic.claude-code
code --install-extension GitHub.copilot
code --install-extension GitHub.copilot-chat
code --install-extension google.gemini-code-assist
code --install-extension AmazonWebServices.aws-toolkit-vscode

# Sur VSCodium (registre OpenVSX)
codium --install-extension Continue.continue
codium --install-extension saoudrizwan.claude-dev
```

9.1.3 Sur VSCodium spécifiquement

VSCodium utilise par défaut le registre **OpenVSX** au lieu du marketplace Microsoft. Toutes les extensions ne sont pas disponibles sur les deux. Pour forcer l'usage d'un `.vsix` téléchargé manuellement :

```
# Télécharger le .vsix depuis Marketplace
codium --install-extension extension.vsix
```

Ou modifier `~/.config/VSCodium/product.json` pour pointer vers le marketplace MS (non officiel mais possible).

9.1.4 Terminal intégré

Tous les IDE basés sur VSCode permettent d'ouvrir un **terminal intégré** (Ctrl+‘) dans lequel n'importe quelle CLI IA fonctionne directement. C'est souvent l'usage le plus efficace : Aider, Plandex, Goose, etc. tournent dans un panneau, et l'IDE détecte automatiquement les modifications fichiers.

9.1.5 MCP (Model Context Protocol)

Plusieurs extensions VSCode commencent à exposer des serveurs MCP (cf. Continue, Cline, Claude Code, Roo). Cela permet à un agent CLI tournant dans un autre terminal d'interroger l'éditeur pour obtenir l'arborescence ouverte, les diagnostics, etc.

9.2 8.2 JetBrains (IntelliJ, PyCharm, GoLand, WebStorm, Rider, RubyMine, CLion, PhpStorm…)

L'intégralité de la suite JetBrains partage le même framework de plugins. Les plugins IA sont donc en général disponibles pour tous les IDE de la suite.

9.2.1 Plugins officiels

Outil	Plugin JetBrains	URL
Claude Code	« Claude Code »	https://plugins.jetbrains.com/plugin/26068-claude-code
Codex CLI	« OpenAI Codex »	Marketplace JetBrains
Gemini CLI	« Gemini Code Assist »	Marketplace JetBrains
Copilot CLI	« GitHub Copilot »	Marketplace JetBrains
Amazon Q	« Amazon Q »	Marketplace JetBrains
Tabnine	« Tabnine »	Marketplace JetBrains
Continue	« Continue »	https://plugins.jetbrains.com/plugin/22707-continue
Cline	« Cline »	Marketplace (depuis 2025)
Cody	« Sourcegraph Cody »	Marketplace JetBrains
Goose	« Goose »	Marketplace (via MCP)
JetBrains AI Assistant (natif)	inclus	—

9.2.2 Installation via CLI

```
# Si JetBrains Toolbox est installé, on peut scripter
# Sinon, installation manuelle via Settings → Plugins
```

9.2.3 Plus la voie du terminal

Les IDE JetBrains intègrent un terminal (View → Tool Windows → Terminal). Toutes les CLI fonctionnent. La fonction **« Run with AI Assistant »** des dernières versions (2024.3+) intègre directement plusieurs LLM.

9.2.4 MCP Server JetBrains

Depuis 2025, JetBrains publie un **serveur MCP officiel** (`jetbrains-mcp-server`) qui expose les capacités de l'IDE (lecture du projet, diagnostics, refactoring) à n'importe quel agent MCP-compatible (Claude Code, Goose, Cline, Cursor Agent, Continue…).

```
# Configuration dans le client MCP de l'agent (ex. Claude Code)
claude mcp add jetbrains -- /chemin/vers/jetbrains-mcp-server
```

9.3 8.3 Neovim / Vim

Neovim est l'IDE des puristes du terminal, et de nombreux plugins existent pour l'IA.

9.3.1 Plugins recommandés

Outil	Plugin Neovim
Claude Code	`coder/claudecode.nvim`
Codex CLI	`johnseth97/codex.nvim`
Aider	`joshuavial/aider.nvim`, `GeorgesAlkhouri/nvim-aider`
Continue	`continuedev/continue.nvim` (officiel)
Copilot	`github/copilot.vim` (officiel), `zbirenbaum/copilot.lua`
CopilotChat	`CopilotC-Nvim/CopilotChat.nvim`
Codeium / Windsurf	`Exafunction/codeium.nvim`
Tabby	`TabbyML/vim-tabby`
Ollama (chat)	`David-Kunz/gen.nvim`, `nomnivore/ollama.nvim`, `oysandvik94/curl.nvim`
Multi-LLM	`olimorris/codecompanion.nvim`, `yetone/avante.nvim`, `David-Kunz/gen.nvim`
llm (Simon Willison)	`Robitx/gp.nvim`

9.3.2 Exemple : configuration minimale avec lazy.nvim

```lua
-- ~/.config/nvim/lua/plugins/ai.lua
return {
  { "olimorris/codecompanion.nvim",
    dependencies = { "nvim-lua/plenary.nvim", "MeanderingProgrammer/render-markdown.nvim" },
    config = function()
      require("codecompanion").setup({
        adapters = {
          anthropic = function() return require("codecompanion.adapters").extend("anthropic", {}) end,
          ollama = function() return require("codecompanion.adapters").extend("ollama", {}) end,
        }
      })
    end
  },
  { "github/copilot.vim" },
  { "yetone/avante.nvim", build = "make" },
}
```

9.3.3 Astuce : tmux + agent CLI

Beaucoup d'utilisateurs préfèrent simplement avoir Aider/Plandex/Goose dans un panneau tmux à côté de Neovim, plutôt qu'un plugin. Cette approche reste la plus stable.

9.4 8.4 Emacs

Emacs dispose de plusieurs paquets pour l'IA :

Outil	Paquet Emacs
Multi-LLM (généraliste)	`gptel` (le plus populaire)
Claude Code	`claude-code.el`
Aider	`aider.el`, `aidermacs`
Copilot	`copilot.el`
Ollama	`ellama`
Khoj	`khoj.el`
llm	`llm.el`

9.4.1 Exemple : configuration `gptel`

```
(use-package gptel
  :ensure t
  :config
  (setq gptel-default-mode 'org-mode
        gptel-model 'claude-sonnet-4-5
        gptel-backend
        (gptel-make-anthropic "Claude"
          :stream t
          :key (getenv "ANTHROPIC_API_KEY"))))
```

9.4.2 Emacs + LLM via tramp

Emacs peut aussi piloter un agent CLI distant via `M-x shell` ou `M-x term`. Combiné à Org-mode et au mode `babel`, cela permet de scripter des appels LLM dans un fichier `.org`.

9.5 8.5 Helix

Helix (éditeur modal écrit en Rust, *post-Vim*) ne dispose pas encore d'écosystème de plugins IA aussi riche que Neovim, mais s'utilise très bien avec **smartcat** :

```
# Sélection puis transformation par sc
:|sc "ajoute des docstrings"
:|sc -p commit-message
```

9.6 8.6 Sublime Text et autres

- **Sublime Text** : `Tabnine`, `Codeium`, `LSP-copilot`, `OpenAI Sublime Text`.
- **Atom / Pulsar** : extensions communautaires (ralenti depuis l'arrêt d'Atom).
- **Eclipse** : Tabnine, Amazon Q, Copilot officiels.
- **Visual Studio (Microsoft)** : Copilot natif, Tabnine, Amazon Q officiels.

9.7 8.7 Recettes types

9.7.1 Workflow A — Aider dans tmux à côté de Neovim

```
# Pane 1 : nvim
nvim .
# Pane 2 (Ctrl+b ") : aider
cd $PWD && aider --architect --model claude-sonnet-4.5
```

Avantages : indépendance totale, latence minimale, pas de plugin à maintenir.

9.7.2 Workflow B — VSCode + Continue + Ollama local

1. Installer Ollama : `curl -fsSL https://ollama.com/install.sh | sh`
2. Tirer un modèle codeur : `ollama pull qwen2.5-coder:14b`
3. Installer Continue : `code --install-extension Continue.continue`
4. Configurer `~/.continue/config.json` :

```
{
  "models": [
    {"title":"Qwen Coder 14B","provider":"ollama","model":"qwen2.5-coder:14b"}
  ],
  "tabAutocompleteModel": {"title":"Qwen Coder","provider":"ollama","model":"qwen2.5-coder:1.5b"}
}
```

5. Aucune donnée ne quitte votre poste.

9.7.3 Workflow C — JetBrains + MCP + Claude Code en terminal externe

1. Démarrer le serveur MCP JetBrains.
2. Dans un terminal séparé : `claude mcp add jetbrains -- /chemin/jetbrains-mcp-server`
3. Lancer `claude` : il dispose maintenant de la connaissance complète du projet ouvert dans IntelliJ.

9.7.4 Workflow D — DevOps : générer un message de commit puis pousser

```
git add -A
git diff --cached | mods "écris un commit conventionnel" | git commit -F-
# ou
oco --yes
git push
```

Chapter 10

Partie IX — Lexique technique

Cette partie définit les termes techniques rencontrés tout au long du guide. L'écosystème de l'IA générative invente du vocabulaire à grande vitesse ; ce lexique est forcément incomplet mais devrait couvrir l'essentiel.

10.1 A

Agent (LLM) — Programme dans lequel un modèle de langage prend des décisions en boucle (planification, exécution d'outils, vérification du résultat) pour accomplir une tâche complexe. Par opposition à un *chat* simple où le LLM se contente de répondre.

Agentique — Adjectif désignant les outils ou systèmes capables de comportement autonome multi-étapes via un agent.

ACI (Agent-Computer Interface) — Couche d'interfaçage entre un LLM et son environnement (filesystem, shell, web). Concept popularisé par SWE-agent.

API compatible OpenAI — API HTTP qui imite les endpoints `/v1/chat/completions`, `/v1/embeddings` etc. de l'API d'OpenAI. Quasi-standard *de facto* (Ollama, vLLM, LM Studio, llamafile, LocalAI… exposent tous une telle API).

Auto-complete (FIM) — Complétion automatique de code par un modèle, typiquement entraîné en *Fill-In-the-Middle* (FIM) pour suggérer le code entre un préfixe et un suffixe.

AWQ — *Activation-aware Weight Quantization*, méthode de quantification 4-bit performante.

10.2 B

Backend (LLM) — Service / runtime exécutant le modèle. Exemples : Ollama, vLLM, llama.cpp, OpenAI cloud.

Bedrock (Amazon) — Service AWS hébergeant des LLM tiers (Anthropic, Meta, Mistral…).

10.3 C

CLAUDE.md — Convention introduite par Claude Code : un fichier markdown placé à la racine d'un projet, automatiquement chargé dans le contexte de l'agent au démarrage. Sert à documenter les conventions du projet pour l'IA.

Computer Use — Capacité d'un modèle à interagir directement avec un OS (souris, clavier, captures d'écran). Anthropic a popularisé cette appellation en 2024.

Contexte (fenêtre de contexte) — Nombre maximum de tokens qu'un LLM peut lire en une fois. Va de 8 K à 2 M tokens selon les modèles.

Conventional Commits — Norme de format des messages de commit (`feat:`, `fix:`, `chore:` …). Fortement utilisée par les générateurs de commit IA.

10.4 D

DSPy — Framework Stanford pour programmer les LLM de manière déclarative (concept de *signature* / *module*).

Diff — Représentation des changements entre deux versions d'un fichier. Format *unified* (`@@`) le plus courant. Les agents codeurs travaillent souvent en générant des diffs.

10.5 E

Embedding — Représentation vectorielle dense d'un texte. Base du RAG.

EnIGMA — Mode de SWE-agent dédié aux Capture-The-Flag de cybersécurité.

10.6 F

Function calling — Capacité d'un LLM à produire un appel de fonction structuré (JSON) plutôt qu'une réponse en langage naturel. Permet la création d'outils.

10.7 G

GGUF — Format de fichier de modèles utilisé par llama.cpp / Ollama / llamafile. Successeur de GGML.

Grounding — Ancrer la réponse d'un LLM dans des faits externes (web search, RAG documentaire) pour limiter les hallucinations.

GPTQ — Méthode de quantification 4-bit pour LLM (alternative à AWQ).

10.8 H

Hallucination — Production par un LLM d'une réponse fausse présentée comme vraie. Risque inhérent aux modèles génératifs.

Hooks (Claude Code) — Mécanisme permettant de déclencher un script avant ou après chaque appel d'outil de l'agent (analogie avec les Git hooks).

10.9 I

Inference (inférence) — Phase où le modèle, déjà entraîné, produit une réponse à partir d'un prompt.

10.10 J

JSON Schema — Description formelle de la structure d'un objet JSON, utilisée pour contraindre les sorties d'un LLM.

10.11 L

LangChain / LangGraph — Framework Python pour construire des applications LLM (LangChain) et des agents stateful (LangGraph).

LangSmith / Langfuse — Plateformes d'observabilité des appels LLM (traces, scores, évaluations).

LiteLLM — Bibliothèque Python qui unifie l'accès à 200+ providers LLM derrière une API OpenAI-like. Utilisée par Aider, OpenHands, Mentat, etc.

LLM (Large Language Model) — Modèle de langage à grande échelle (GPT, Claude, Llama, Gemini…).

LoRA / QLoRA — Méthodes d'adaptation efficace de LLM (fine-tuning paramétrique léger).

10.12 M

MCP (Model Context Protocol) — Protocole ouvert créé par Anthropic en 2024 pour standardiser la communication entre un LLM et ses outils/données. Permet à un agent (Claude Code, Goose, Cursor…) de se connecter à des serveurs MCP exposant des fonctionnalités (filesystem, GitHub, Postgres, JetBrains…).

Modelfile (Ollama) — Fichier de configuration d'un modèle dérivé (équivalent d'un Dockerfile pour les LLM).

Multimodal — Modèle qui accepte plusieurs types d'entrée (texte + image + audio…).

10.13 O

OCI Artifacts — Standard de stockage de blobs (modèles, configurations) dans des registres de containers. Utilisé par RamaLama.

Ollama Hub — Registre public de modèles Ollama.

10.14 P

PagedAttention — Technique de gestion de la mémoire KV-cache de vLLM, inspirée de la mémoire virtuelle des OS.

Pattern (fabric) — Prompt système pré-écrit et nommé, application d'une recette d'IA reproductible.

Prompt — Texte d'entrée envoyé à un LLM.

Prompt engineering — Discipline de l'écriture de prompts efficaces.

Prompt injection — Attaque consistant à insérer des instructions malicieuses dans une entrée que le LLM va traiter (ex. dans un document que l'agent lit).

10.15 Q

Quantization — Réduction de la précision des poids d'un modèle (16 bit → 8/4/3/2 bit) pour réduire la taille et accélérer l'inférence, au prix d'une légère perte de qualité.

10.16 R

RAG (Retrieval-Augmented Generation) — Technique combinant recherche documentaire (vector store) et LLM : la requête est d'abord utilisée pour retrouver des chunks pertinents, qui sont injectés dans le contexte du LLM avant la génération.

Refcard — Carte de référence : liste concise des commandes/options d'un outil.

REPL — *Read-Eval-Print Loop*, boucle interactive (saisir une expression, voir le résultat, recommencer).

Roo / Cline / Continue Modes — Modes de travail différenciés (Plan, Code, Architect, Ask, Debug) proposés par certains agents.

10.17 S

Sandbox — Environnement isolé où l'agent peut exécuter du code sans danger pour le système hôte. Utilisé par Codex CLI (Landlock+seccomp), OpenHands (Docker), Goose, etc.

SDK — *Software Development Kit*, ensemble d'outils/bibliothèques pour développer.

Skill (Claude) — Plugin installable côté Claude (UI ou CLI) qui apporte une capacité ciblée.

Slash command — Commande commençant par `/` à l'intérieur d'un agent CLI (ex. `/help`, `/clear`).

SOP (Standard Operating Procedure) — Procédure standardisée. Concept utilisé par MetaGPT pour décrire les rôles de ses agents.

SWE-bench — Benchmark de résolution autonome de bugs GitHub. Métrique de référence pour comparer les agents codeurs.

10.18 T

Tokens — Unités élémentaires sur lesquelles les LLM travaillent (pseudo-mots).

Tool / Tool calling / Function calling — Mécanisme par lequel un LLM peut appeler une fonction externe (filesystem, web, etc.).

TUI (Text User Interface) — Interface utilisateur en mode texte (ncurses, Bubble Tea, Ratatui…).

10.19 V

Vector store — Base de données spécialisée dans la recherche de vecteurs proches (Chroma, Qdrant, Pinecone, Weaviate…). Cœur du RAG.

Vertex AI — Plateforme Google Cloud pour les modèles IA, notamment Gemini.

vLLM — Moteur d'inférence haute performance basé sur PagedAttention.

10.20 W

Whisper — Modèle d'OpenAI pour la transcription audio en texte. `whisper.cpp` en est l'implémentation C++.

10.21 Y

YOLO mode — Mode d'agent dans lequel l'utilisateur autorise toutes les actions sans confirmation (« You Only Live Once »). Présent dans Gemini CLI, Warp, Cline, etc.

10.22 Z

Zero retention — Engagement contractuel d'un fournisseur LLM cloud à ne pas conserver les requêtes ni les utiliser pour l'entraînement.

Chapter 11

Partie X — Liens et ressources complémentaires

Cette dernière partie rassemble des liens utiles pour aller plus loin : sites officiels, dépôts GitHub, listes communautaires, blogs, podcasts et benchmarks.

11.1 10.1 Sites officiels et dépôts GitHub

11.1.1 CLI propriétaires majeurs

- Anthropic Claude Code — https://www.claude.com/product/claude-code · https://github.com/anthropics/claude-code
- OpenAI Codex CLI — https://developers.openai.com/codex/cli · https://github.com/openai/codex
- Google Gemini CLI — https://cloud.google.com/gemini/docs/codeassist/gemini-cli · https://github.com/google-gemini/gemini-cli
- GitHub Copilot CLI — https://github.com/features/copilot · https://github.com/github/gh-copilot
- Cursor CLI — https://cursor.com/cli · https://docs.cursor.com/cli
- Sourcegraph Cody — https://sourcegraph.com/cody · https://github.com/sourcegraph/cody
- Amazon Q Developer — https://aws.amazon.com/q/developer/
- Warp Terminal — https://www.warp.dev
- Tabnine — https://www.tabnine.com
- JetBrains AI Assistant — https://www.jetbrains.com/ai/
- Replit Agent — https://replit.com
- Codeium / Windsurf — https://codeium.com/windsurf

11.1.2 Agents codeurs OSS

- Aider — https://aider.chat · https://github.com/Aider-AI/aider
- Plandex — https://plandex.ai · https://github.com/plandex-ai/plandex
- Goose — https://block.github.io/goose/ · https://github.com/block/goose
- OpenHands — https://www.all-hands.dev · https://github.com/All-Hands-AI/OpenHands
- Cline — https://cline.bot · https://github.com/cline/cline
- Continue — https://continue.dev · https://github.com/continuedev/continue
- Open Interpreter — https://openinterpreter.com · https://github.com/OpenInterpreter/open-interpreter
- gpt-engineer — https://github.com/AntonOsika/gpt-engineer
- gptme — https://gptme.org · https://github.com/gptme/gptme
- SWE-agent — https://swe-agent.com · https://github.com/SWE-agent/SWE-agent
- Mentat — https://github.com/AbanteAI/mentat
- Smol Developer — https://github.com/smol-ai/developer
- Roo Code — https://github.com/RooVetGit/Roo-Code
- Crush (Charm) — https://github.com/charmbracelet/crush

- Forge — https://github.com/antinomyhq/forge
- Devon — https://github.com/entropy-research/Devon
- GPT Pilot / Pythagora — https://github.com/Pythagora-io/gpt-pilot
- Sweep AI — https://github.com/sweepai/sweep
- Khoj — https://khoj.dev · https://github.com/khoj-ai/khoj

11.1.3 CLI LLM génériques

- llm (Simon Willison) — https://llm.datasette.io · https://github.com/simonw/llm
- ShellGPT (sgpt) — https://github.com/TheR1D/shell_gpt
- aichat — https://github.com/sigoden/aichat
- mods — https://github.com/charmbracelet/mods
- tgpt — https://github.com/aandrew-me/tgpt
- fabric — https://github.com/danielmiessler/fabric
- gptscript — https://github.com/gptscript-ai/gptscript
- chatgpt-cli — https://github.com/kardolus/chatgpt-cli
- gpt-cli — https://github.com/kharvd/gpt-cli
- smartcat — https://github.com/efugier/smartcat
- chatblade — https://github.com/npiv/chatblade
- gorilla-cli — https://gorilla.cs.berkeley.edu · https://github.com/gorilla-llm/gorilla-cli
- shai (OVHcloud) — https://github.com/ovh/shai
- wrangler ai — https://github.com/cloudflare/workers-sdk

11.1.4 Runners locaux

- Ollama — https://ollama.com · https://github.com/ollama/ollama
- llama.cpp — https://github.com/ggml-org/llama.cpp
- LM Studio — https://lmstudio.ai · https://github.com/lmstudio-ai/lms
- llamafile — https://llamafile.ai · https://github.com/Mozilla-Ocho/llamafile
- vLLM — https://docs.vllm.ai · https://github.com/vllm-project/vllm
- GPT4All — https://gpt4all.io · https://github.com/nomic-ai/gpt4all
- LocalAI — https://localai.io · https://github.com/mudler/LocalAI
- Jan / Cortex — https://jan.ai · https://github.com/janhq/jan
- RamaLama — https://github.com/containers/ramalama
- KoboldCpp — https://github.com/LostRuins/koboldcpp
- text-generation-webui (Oobabooga) — https://github.com/oobabooga/text-generation-webui
- mlx-lm (Apple) — https://github.com/ml-explore/mlx-examples
- whisper.cpp — https://github.com/ggml-org/whisper.cpp
- piper TTS — https://github.com/rhasspy/piper
- stable-diffusion.cpp — https://github.com/leejet/stable-diffusion.cpp
- Coqui TTS — https://github.com/idiap/coqui-ai-TTS

11.1.5 Outils spécialisés

- aicommits — https://github.com/Nutlope/aicommits
- opencommit — https://github.com/di-sukharev/opencommit
- gptcommit — https://github.com/zurawiki/gptcommit
- AI Shell — https://github.com/BuilderIO/ai-shell
- Butterfish — https://butterfi.sh · https://github.com/bakks/butterfish
- shell-ai (q) — https://github.com/ricklamers/shell-ai
- Pieces CLI — https://pieces.app · https://github.com/pieces-app/cli-agent
- promptfoo — https://www.promptfoo.dev · https://github.com/promptfoo/promptfoo
- Tabby — https://tabby.tabbyml.com · https://github.com/TabbyML/tabby
- ata — https://github.com/rikhuijzer/ata

11.1.6 Frameworks d'agents

- AutoGPT — https://agpt.co · https://github.com/Significant-Gravitas/AutoGPT
- BabyAGI — https://github.com/yoheinakajima/babyagi
- CrewAI — https://crewai.com · https://github.com/crewAIInc/crewAI
- MetaGPT — https://www.deepwisdom.ai · https://github.com/geekan/MetaGPT
- SuperAGI — https://superagi.com · https://github.com/TransformerOptimus/SuperAGI
- AgentGPT — https://github.com/reworkd/AgentGPT
- LangChain — https://python.langchain.com · https://github.com/langchain-ai/langchain
- LangGraph — https://langchain-ai.github.io/langgraph · https://github.com/langchain-ai/langgraph
- Haystack — https://haystack.deepset.ai · https://github.com/deepset-ai/haystack
- Hayhooks — https://github.com/deepset-ai/hayhooks
- Letta (ex-MemGPT) — https://letta.com · https://github.com/letta-ai/letta
- Agno (ex-Phidata) — https://agno.com · https://github.com/agno-agi/agno
- Pydantic AI — https://ai.pydantic.dev · https://github.com/pydantic/pydantic-ai
- Semantic Kernel — https://learn.microsoft.com/semantic-kernel · https://github.com/microsoft/semantic-kernel
- DSPy — https://github.com/stanfordnlp/dspy
- AutoGen — https://github.com/microsoft/autogen
- OpenAI Swarm — https://github.com/openai/swarm
- Camel-AI — https://github.com/camel-ai/camel

11.2 10.2 Modèles et fournisseurs LLM

- Anthropic — https://www.anthropic.com
- OpenAI — https://openai.com
- Google AI — https://ai.google
- Mistral AI (France) — https://mistral.ai
- Cohere — https://cohere.com
- Groq — https://groq.com
- Together AI — https://together.ai
- DeepSeek — https://deepseek.com
- Qwen (Alibaba) — https://qwenlm.com
- Hugging Face — https://huggingface.co
- OpenRouter — https://openrouter.ai
- Fireworks AI — https://fireworks.ai
- Replicate — https://replicate.com
- Perplexity — https://www.perplexity.ai
- xAI Grok — https://x.ai
- Inception (Mercury) — https://inceptionlabs.ai

11.3 10.3 Spécifications et standards

- **MCP (Model Context Protocol)** — https://modelcontextprotocol.io · https://github.com/modelcontextprotocol
- **OpenAI API spec (de facto standard)** — https://platform.openai.com/docs/api-reference
- **GGUF** — https://github.com/ggerganov/ggml/blob/master/docs/gguf.md
- **OCI Artifacts** — https://github.com/opencontainers/artifacts

11.4 10.4 Benchmarks publics

- **SWE-bench / SWE-bench Verified** — https://www.swebench.com
- **Aider polyglot benchmark** — https://aider.chat/docs/leaderboards/
- **LMSYS Chatbot Arena** — https://chat.lmsys.org

- **Open LLM Leaderboard (HF)** — https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard
- **HumanEval** — https://github.com/openai/human-eval
- **MMLU / MMLU-Pro** — https://github.com/hendrycks/test
- **LiveCodeBench** — https://livecodebench.github.io
- **GAIA** — https://huggingface.co/spaces/gaia-benchmark/leaderboard
- **TerminalBench** — https://github.com/laude-institute/terminal-bench

11.5 10.5 Listes / catalogues communautaires

- **awesome-llm** — https://github.com/Hannibal046/Awesome-LLM
- **awesome-ai-cli** — https://github.com/topics/ai-cli
- **awesome-mcp-servers** — https://github.com/punkpeye/awesome-mcp-servers
- **awesome-llm-apps** — https://github.com/Shubhamsaboo/awesome-llm-apps
- **awesome-claude-code** — https://github.com/hesreallyhim/awesome-claude-code
- **awesome-codex** — https://github.com/topics/openai-codex
- **awesome-langchain** — https://github.com/kyrolabs/awesome-langchain
- **awesome-local-llm** — https://github.com/vince-lam/awesome-local-llms

11.6 10.6 Communauté francophone

- **LinuxFr.org** — https://linuxfr.org (rubrique IA)
- **Linux Maine** (Thierry Gayet) — https://linuxmaine.org
- **Developpez.com — Forum IA** — https://www.developpez.net/forums
- **Forum Ubuntu-fr** — https://forum.ubuntu-fr.org
- **r/france /r/programmingfr** sur Reddit
- **AI Builders France** — communautés Discord/Slack

11.7 10.7 Documentation et apprentissage

- **Anthropic Cookbook** — https://github.com/anthropics/anthropic-cookbook
- **OpenAI Cookbook** — https://github.com/openai/openai-cookbook
- **Google Gemini Cookbook** — https://github.com/google-gemini/cookbook
- **DeepLearning.AI courses** — https://www.deeplearning.ai
- **Simon Willison's blog** — https://simonwillison.net (LLM weekly)
- **Latent Space (podcast)** — https://www.latent.space
- **Lilian Weng's blog (OpenAI)** — https://lilianweng.github.io
- **Hamel Husain's notes** — https://hamel.dev
- **Eugene Yan's blog** — https://eugeneyan.com

11.8 10.8 Newsletters et fils d'actualité

- **The Batch (Andrew Ng)** — https://www.deeplearning.ai/the-batch/
- **Import AI (Jack Clark)** — https://importai.substack.com
- **TLDR AI** — https://tldr.tech/ai
- **Last Week in AI** — https://lastweekin.ai
- **AInauts (FR)** — https://ainauts.fr

11.9 10.9 Outils complémentaires

- **AnythingLLM** — https://github.com/Mintplex-Labs/anything-llm
- **NextChat (anciennement ChatGPT-Next-Web)** — https://github.com/ChatGPTNextWeb/NextChat
- **MindsDB** — https://github.com/mindsdb/mindsdb
- **Flowise** — https://github.com/FlowiseAI/Flowise

- **n8n (workflows IA)** — https://github.com/n8n-io/n8n
- **SillyTavern (chat IA RP)** — https://github.com/SillyTavern/SillyTavern

11.10 10.10 Sites officiels Anthropic / OpenAI / Google détaillés

- Anthropic Console — https://console.anthropic.com
- Anthropic API docs — https://docs.anthropic.com
- OpenAI Platform — https://platform.openai.com
- Google AI Studio — https://aistudio.google.com
- Vertex AI — https://cloud.google.com/vertex-ai
- AWS Bedrock — https://aws.amazon.com/bedrock/
- Azure OpenAI — https://azure.microsoft.com/fr-fr/products/ai-services/openai-service

Chapter 12

Conclusion

L'écosystème des CLI IA est probablement l'un des plus dynamiques de l'open-source actuel : plus de quarante outils significatifs sont apparus en moins de trois ans, et le rythme s'accélère. On y trouve :

- une **professionnalisation rapide** des agents codeurs (Claude Code, Codex, Cursor, Aider, Plandex, Goose…) qui arrivent à un niveau de maturité comparable à celui des IDE traditionnels au début des années 2010 ;
- une **explosion de l'inférence locale** (Ollama, llama.cpp, LM Studio, llamafile, vLLM…), qui rend désormais accessible sur un poste personnel ce qui nécessitait des datacenters il y a deux ans ;
- une **standardisation autour de MCP**, qui commence à jouer le rôle qu'a joué LSP pour les éditeurs de code ;
- la **persistance des CLI Unix-style** (`llm`, `mods`, `aichat`, `fabric`, `smartcat`) qui s'intègrent naturellement aux pipelines shell et restent l'option la plus puissante pour scripter l'IA dans des environnements DevOps.

Le choix d'un outil dépend toujours du **cas d'usage** (refactoring, génération, RAG, commits…), du **modèle économique** (cloud payant, OSS, self-hosting) et des **exigences de confidentialité** (données sensibles, souveraineté EU).

Quelle que soit la voie choisie, retenons trois principes :

1. **Toujours sandboxer** un agent autonome (Docker, Landlock, conteneur, VM).
2. **Toujours versionner** ses changements (Aider/Plandex/Goose font des commits Git automatiques pour cette raison).
3. **Toujours relire** ce que produit l'IA avant de pousser en production : les hallucinations existent, même chez les meilleurs modèles.

Bonne exploration et bons hacks !

— *Thierry Gayet* — linuxmaine.org

Chapter 13

Annexe — Note méthodologique

Ce guide a été compilé à partir d'une connaissance encyclopédique des outils CLI IA disponibles fin 2025 / début 2026. La majorité des informations (noms de binaires, commandes d'installation, sous-commandes, options, intégrations IDE) sont stables et largement documentées dans les dépôts officiels.

Précisions sur les versions : les numéros de version exacts à la date de mai 2026 doivent être confirmés sur les pages *Releases* GitHub des projets concernés avant toute installation. La trajectoire de release de chaque outil est très variable (semaine, mois, trimestre). Les commandes d'installation, en revanche, sont presque toujours stables sur plusieurs cycles.

Points à revérifier en priorité :

- Outils avec des homonymes (`shai`, `hai`, `ata`, `q`) : confirmer l'éditeur et le dépôt visé.
- Outils en pleine évolution (Claude Code, Codex, Gemini CLI) : les sous-commandes et flags évoluent presque mensuellement.
- Tarifications cloud : changent fréquemment, vérifier les pages officielles avant communication.

Contributions : si vous repérez des erreurs, des outils manquants ou des informations obsolètes, n'hésitez pas à contacter l'auteur via https://linuxmaine.org.