# Codex, OpenCode, Pi или MiMo Code: выбор coding agent для OpenAI models

| | |
|---|---|
| **Date:** | 13 июля 2026 года |
| **Query:** | Какой coding agent выбрать для работы с OpenAI models? |
| **Sources:** | 35 |
| **Citation coverage:** | 95% |

## Аннотация

Этот обзор отвечает на практический вопрос: какой coding agent выбрать для работы прежде всего с OpenAI models — официальный Codex, открытый OpenCode, минимальный Pi, новый MiMo Code или другой инструмент. Срез зафиксирован на 13 июля 2026 года. Исследование опирается на официальную документацию продуктов, академические работы о benchmarks, три класса исследований производительности и публикации о безопасности agentic systems.

Главный вывод: **для большинства разработчиков, которые осознанно выбирают OpenAI models, разумный default — Codex**. Он лучше других связывает актуальные OpenAI models с локальной работой, IDE, desktop и cloud, а также предоставляет готовые sandbox, approvals и network controls.[1] [2] **OpenCode** стоит выбрать, если важнее открытый код, provider portability и возможность уйти на local models. **Pi** подходит тем, кто хочет короткий, наблюдаемый и программируемый harness и готов самостоятельно построить permission model. **MiMo Code** интересен persistent memory и orchestration, но пока слишком молод, а его сравнительные преимущества подтверждены главным образом vendor-run tests.[8] [12] [14]

Ни один публичный leaderboard не даёт достаточных оснований объявить один продукт лучшим. Результат coding agent — это не только модель, а произведение `model × harness × context policy × tools × permissions × budget × environment`. Для команды окончательное решение должен принимать короткий private bake-off на собственных задачах, с учётом стоимости принятого изменения, времени review и частоты переделок.[21] [24] [25]

## Введение: что именно мы выбираем

Слово «agent» скрывает два разных слоя. **Model** генерирует рассуждение, текст и tool calls. **Agent harness** решает, какие файлы попадут в контекст, какие tools доступны, когда выполнять shell command, как сжимать историю, когда просить подтверждение, как запускать тесты и что считать завершением. Codex, OpenCode, Pi и MiMo Code могут обращаться к OpenAI models, но дают им разные рабочие среды. Поэтому вопрос «какой agent лучше работает с OpenAI» нельзя свести к сравнению GPT с Claude или MiMo.[9] [12][14]

В обзор включены терминальные и облачные agents, способные читать repository, изменять несколько файлов, запускать команды и итеративно проверять результат. Inline autocomplete рассматривается только как соседний режим работы. Не сравниваются обычные chat-интерфейсы и конструкторы общих business agents: они решают другую задачу.

Критерии выбора:

1. качество связки с OpenAI models;
2. управляемость контекста и долгих задач;
3. sandbox, approvals и работа с недоверенным содержимым;
4. provider portability и local inference;
5. зрелость, расширяемость и интеграции;
6. полная стоимость принятого изменения, а не цена подписки;
7. пригодность для локальной работы, IDE, CI и cloud delegation.

## Короткий ответ

| Сценарий | Рекомендация | Почему | Главная оговорка |
|---|---|---|---|
| OpenAI models как основной выбор | **Codex** | First-party model support, local/IDE/desktop/cloud, sandbox и approvals в одной системе[1][2] | Vendor lock-in, переменная стоимость, code/context обрабатывает OpenAI |
| Нужны OpenAI и другие providers | **OpenCode** | MIT, 75+ providers, local models, plan/build agents, permissions, MCP и LSP[8][9][10] | Качество зависит от конкретной пары model/harness; больше настроек и surface area |
| Нужен собственный минимальный harness | **Pi** | MIT, прозрачное ядро, extensions, SDK/RPC, переключение providers[12][13] | Нет встроенных permission prompts; безопасная среда — обязанность пользователя |
| Нужны memory и сложная orchestration | **MiMo Code**, только pilot | Persistent memory, checkpoints, compose workflows, subagents и goal judge[14][15] | Выпущен 15 июня 2026 года; benchmark пока vendor-run |
| Главный интерфейс — IDE | **Cursor** или **Cline** | Cursor даёт законченный proprietary UX; Cline — open-source BYOK в IDE[17] [18] | Cursor проксирует AI traffic; Cline требует собрать provider и policy |
| Issue → PR внутри GitHub | **GitHub Copilot coding agent** | Ephemeral environment и GitHub-native workflow[20] | Это не универсальный локальный terminal agent; расходуются agent/Actions resources |
| Простое git-centric pair programming | **Aider** | Repo map, architect/editor mode и явная git-модель[19] | По autonomous features и темпу развития уступает новым agents |

Если нужен один ответ без дополнительных условий: **начните с Codex, оставьте OpenCode вторым harness для portability, а Pi и MiMo Code оценивайте как специализирован-**

**ные варианты**. Это не утверждение, что Codex всегда генерирует лучший patch. Это вывод о суммарном риске внедрения для пользователя OpenAI models.

## Почему Codex — основной default для OpenAI models

Codex — единственный вариант в сравнении, где provider модели, model family и agent surfaces принадлежат одному поставщику. Официальная документация объединяет CLI для локального repository, IDE extension, desktop app и cloud tasks. CLI читает и меняет файлы, запускает установленные tools и имеет non-interactive `codex exec`; cloud запускает задачи параллельно в изолированных environments и возвращает summary с diff перед созданием PR.[1][3]

> "Inspect, edit, and run code from your terminal."[1]

Эта вертикальная интеграция важна не как бренд, а как снижение числа неподтверждённых предположений. В model-neutral harness разработчики должны проверить совместимость reasoning levels, tool-call schema, caching, compaction и обработку ошибок для каждой новой модели. Codex может менять harness вместе с OpenAI model release. На 13 июля 2026 года текущим семейством OpenAI является GPT-5.6: Sol для наиболее сложных задач, Terra как баланс качества и цены, Luna для более дешёвых и быстрых workloads.[4] Этот список быстро устаревает, но first-party путь уменьшает задержку между выпуском модели и корректной поддержкой в agent.

Вторая сильная сторона — несколько режимов исполнения без смены продукта. Локальный режим подходит для интерактивной работы; worktree изолирует параллельную ветку; cloud удобен для долгих задач и issue-to-PR delegation. Это позволяет не заставлять один интерфейс обслуживать все workflows. Долгий migration можно отдать cloud task, а неоднозначную правку вести локально с частыми остановками и review.[3]

Третья причина — готовая security-модель. По умолчанию local Codex работает без network, применяет OS-enforced sandbox, ограничивает запись workspace и использует approvals для выхода за границы. В cloud setup phase может получить зависимости, после чего agent phase работает offline, если пользователь отдельно не разрешил интернет; secrets из setup удаляются до agent phase.[2]

> "By default, the agent runs with network access turned off."[2]

Эти механизмы не делают Codex безопасным автоматически. `danger-full-access` снимает значимую часть границ, а `--yolo` отключает sandbox и approvals. Live web и network увеличивают поверхность indirect prompt injection и exfiltration. Правильный вывод из документации — не «Codex безопасен», а «Codex даёт готовые controls, которые можно включить и централизованно описать».[2]

### Ограничения Codex

Главный функциональный минус — provider lock-in. OpenCode и Pi позволяют менять OpenAI на Anthropic, Google, Bedrock, OpenRouter или local endpoint в пределах одного интерфейса. Codex проектируется вокруг OpenAI models. Если организация требует on-prem inference или хочет ежедневно выбирать лучший model/provider по цене, first-party оптимизация превращается в ограничение.

Второй минус — непредсказуемая стоимость интенсивного agentic workflow. Codex включён в несколько ChatGPT plans, но limits различаются; дополнительное использование переведено на token-based credits. Официальная оценка OpenAI для активного использования — примерно 100–200 долларов на разработчика в месяц со значительной дисперсией из-за модели, parallel instances, automations и fast mode.[5] Это полезный ориентир, а не фиксированный тариф. Длинный context, высокий reasoning effort и несколько параллельных agents могут увеличить расход незаметно для пользователя.

Третий минус — data governance. Для Business, Enterprise, Edu и API inputs/outputs не используются для training по умолчанию. Для индивидуальных планов требуется проверить opt-out; настройка, относящаяся к полному Codex environment, отделена от обычного chat control.[6] Следовательно, «у меня Plus, значит закрытый repository автоматически имеет enterprise policy» — неверная логика.

Четвёртый минус — cloud требует подготовки. Нужно подключить GitHub, описать dependencies, tools, environment variables и secrets, а после выполнения проверить diff и tests. Cloud delegation уменьшает время ожидания человека, но не отменяет review. В официальной security evaluation предыдущего GPT-5.2-Codex защита от destructive actions также не была идеальной; эту цифру нельзя переносить в production failure rate, но она опровергает идею безошибочного автономного исполнителя.[3][7]

## OpenCode: лучший открытый default

OpenCode — наиболее прямой open-source конкурент Codex для пользователя, которому нужны OpenAI models без обязательной привязки к OpenAI. Проект распространяется под MIT, активно развивается и предоставляет terminal UI, desktop beta, built-in build/plan agents и subagent.[8] Документация заявляет более 75 providers через AI SDK и Models.dev, включая OpenAI, Azure OpenAI, Bedrock, OpenRouter, Ollama и другие local endpoints.[9]

Сильная сторона OpenCode — не «более умный agent», а **переносимость рабочего процесса**. Можно начать с OpenAI, затем сравнить другую модель или local deployment, не меняя правила, agents и привычки команды. Это особенно ценно, когда provider policy, региональные ограничения или цена важнее максимального результата одного first-party stack.

OpenCode не требует выбирать между гибкостью и базовой безопасностью. Permission rules поддерживают `allow`, `ask` и `deny` по tools и paths; plan agent по умолчанию запрещает edits, а build agent предназначен для выполнения.[10] Встроены MCP, LSP, skills, plugins и custom tools. Для enterprise доступен central configuration и режим, в котором OpenCode обращается к provider напрямую, хотя точный data path всё равно нужно проверять для выбранного provider и optional OpenCode services.[11]

Цена этой полноты — большая поверхность конфигурации. Provider-neutral harness должен адаптироваться к несовпадающим APIs и model behaviors. Одна модель может хорошо работать с tool schema и compaction OpenCode, другая — хуже. Поддержка 75 providers означает широту, но не гарантирует одинаковое качество каждого сочетания. Поэтому OpenCode следует оценивать с конкретной OpenAI model, теми же reasoning limits и тем же task budget, что Codex.

Отдельно важно различать open-source client и private inference. Если OpenCode вызывает OpenAI API, code/context покидает машину и обрабатывается OpenAI. Local client становится local/private stack только вместе с local model и контролируемым network egress. Открытый исходный код облегчает аудит, но не отменяет prompt injection, вредоносный plugin или ошибочный shell command.

**Рекомендация:** OpenCode — лучший второй инструмент рядом с Codex и лучший первый выбор, если multi-provider является обязательным требованием. Для организации это также полезная страховка от model/provider lock-in.

## Pi: минимальный harness как инженерный конструктор

Pi следует оценивать не как урезанный OpenCode, а как другой продуктовый тезис. Его ядро намеренно минимально, а extensions могут добавить custom tools, compaction, subagents, plan mode, MCP, permission gates, sandbox execution и UI. Проект имеет MIT license, provider abstraction, SDK и RPC; историю можно встраивать в собственные процессы, а не только использовать через TUI.[12]

> "Pi is aggressively extensible so it doesn't have to dictate your workflow."[12]

Для работы с OpenAI это даёт два преимущества. Во-первых, developer видит меньше скрытой orchestration и может точнее контролировать context. Во-вторых, Pi можно использовать как библиотеку: собрать узкого агента для CI, internal CLI или воспроизводимого workflow. Автор Pi связывает качество coding agent прежде всего с context engineering и наблюдаемостью, а не с количеством встроенных features.[13]

Но минимализм переносит ответственность на пользователя. В Pi нет built-in permission popups; официальная рекомендация — container или собственный confirmation extension. Нет built-in MCP, plan mode и subagents: всё это возможно, но должно быть спроектировано или установлено отдельно.[12] Packages и extensions выполняют arbitrary code с полным доступом, поэтому их нужно проверять и закреплять по версии.

Pi разумно выбрать в трёх случаях:

- команда строит собственный agent runtime и хочет SDK/RPC;
- разработчик ценит прозрачность context больше готовых функций;
- есть обязательный container/sandbox layer и инженер, владеющий extensions.

Pi не следует выбирать только потому, что он «лёгкий». Если пользователю нужны готовые permissions, MCP, plan mode и subagents, самостоятельная сборка этих механизмов увеличит совокупную сложность. Для большинства пользователей OpenAI models Codex или OpenCode дают более безопасный старт.

## MiMo Code: сильная идея, но пока эксперимент

Название MiMo обозначает несколько разных сущностей. MiMo-V2.5 и MiMo-V2.5-Pro — модели Xiaomi; Xiaomi MiMo API и Token Plan — hosted inference; **MiMo Code** — отдельный terminal agent. Xiaomi выпустила MiMo Code 15 июня 2026 года как secondary development поверх OpenCode под MIT, дополнив его persistent memory, checkpoints, task tree, parallel subagents, compose workflows и goal judge.[14][15]

Persistent memory — наиболее содержательная инновация MiMo Code. Отдельный subagent сохраняет checkpoint, context reconstruction собирает новую сессию из project memory, progress и последних сообщений, а `/dream` периодически сжимает накопленные сведения. Compose mode задаёт фазы design, implementation, testing и review. Для long-horizon tasks это потенциально уменьшает «амнезию» после compaction.[14][15]

Однако MiMo Code пока нельзя рекомендовать как основной harness для OpenAI models. Текущая документация гарантирует custom OpenAI-compatible provider, но не даёт столь же ясного first-party пути ChatGPT/Codex OAuth, как Codex, OpenCode или Pi. Главная оптимизация MiMo Code заявлена для MiMo models. Использование OpenAI возможно, но это не основной проверенный сценарий продукта.[15]

Xiaomi приводит controlled comparison: при одинаковой MiMo model MiMo Code якобы получил 62% против 57% у Claude Code на SWE-Bench Pro и 73% против 68% на Terminal-Bench 2.[14] Это правильнее, чем сравнивать разные модели, но остаётся vendor benchmark: опубликованная страница не предоставляет независимый replication package и полное описание budget, prompts и task exclusions. Результат заслуживает проверки, но не доказывает превосходство harness.

Есть и юридическая оговорка. Repository указывает MIT, но содержит отдельные use restrictions для military, malicious cyber и autonomous high-risk actions.[15] Enterprise legal review должен учитывать оба документа, а не останавливаться на строке «MIT».

**Рекомендация:** MiMo Code подходит для pilot на длинных автономных задачах, особенно с MiMo models. Для повседневной работы с OpenAI models сначала нужен собственный bake-off против Codex и OpenCode. Месяц существования продукта — недостаточная история для консервативного default.

## Другие релевантные решения

### Claude Code

Claude Code — сильный законченный terminal agent с permissions, sandbox и cloud execution, но его естественная связка — Claude. Он полезен как контрольная группа в bake-off: если задача выбирает лучший итоговый stack, а не обязательно OpenAI, исключать Claude Code нельзя. Если требование — использовать OpenAI models, попытка протянуть их через чужой proprietary harness лишает продукт его основной интеграции и не является рациональным default.[16]

### Cursor и Cline

Cursor подходит разработчикам, для которых главным интерфейсом остаётся AI-native IDE. Он объединяет autocomplete, multi-file agent и cloud features, но AI requests проходят через Cursor infrastructure; Privacy Mode регулирует retention/training, а не превращает hosted model в local inference.[17] Cline — открытая Apache-2.0 альтернатива с BYOK, OpenAI, OpenRouter, Bedrock, Vertex, Ollama и LM Studio. Это разумный выбор для IDE-first команды, готовой самостоятельно определить provider, approvals и billing.[18]

### Aider

Aider остаётся понятным git-centric pair programmer. Его repo map ранжирует важные symbols, architect mode разделяет reasoning и editing models, а изменения естественно укладываются в git workflow.[19] Но последняя официальная версия датирована августом 2025 года, а современные competitors быстрее развивают cloud delegation, subagents и policy controls. Aider стоит выбирать за простоту и предсказуемую pair-programming модель, а не за максимальную автономность.

### GitHub Copilot coding agent

GitHub Copilot coding agent естественен для workflow `issue → ephemeral environment → pull request`. GitHub описывает firewall, secret scanning, CodeQL и ограниченную среду, но также перечисляет риски prompt injection и ограничения firewall.[20] Это лучший кандидат, когда единицей работы является GitHub issue, а не локальная интерактивная сессия. Цена seat не

отражает полный расход: agent tasks используют premium requests/AI credits и GitHub Actions resources.

Roo Code не включён в рекомендации: repository был архивирован после закрытия продукта в мае 2026 года. Windsurf остаётся релевантным proprietary AI IDE, но для задачи «OpenAI models через контролируемый harness» не даёт уникального преимущества перед Cursor, Cline или OpenCode.

## Что действительно говорят benchmarks

### SWE-bench не является рейтингом повседневных agents

SWE-bench Verified содержит 500 вручную проверенных задач из 12 Python repositories. Он полезен для сравнения способности исправлять issue в воспроизводимом environment, но выборка не представляет все языки, greenfield work, design, review и эксплуатационные задачи.[21] Статический публичный набор также подвержен optimization и contamination.

Свежие datasets показывают, насколько опасно принимать абсолютный score за универсальную способность. В SWE-bench-Live связка OpenHands с Claude 3.7 Sonnet получила 43,20% на Verified и 19,25% на свежем Live. Задачи из знакомых benchmark repositories решались чаще, чем из новых repositories.[22] Это согласуется с overfitting или implicit optimization, хотя не доказывает единственную причину. SWE-rebench строит поток свежих задач и также обнаруживает, что результаты части моделей на Verified могут быть завышены contamination.[23]

В июле 2026 года OpenAI отдельно заявила, что SWE-bench Verified перестал давать надёжный сигнал из-за design и contamination issues, и предложила более длинные и реалистичные SWE-bench Pro tasks.[34] Поскольку это позиция заинтересованного vendor, её нужно читать вместе с независимыми SWE-bench-Live и SWE-rebench, а не вместо них.

### Terminal-Bench измеряет stack, а не одну модель

Terminal-Bench 2.0 содержит 89 terminal tasks и сравнивает model co scaffold. Авторы выбирают для каждой модели scaffold, который даёт лучший результат: например, Codex CLI с GPT-5.2 получил 63%, Terminus 2 с Claude Opus 4.5 — 58%, Terminus 2 с Gemini 3 Pro — 57%.[24] Это полезная верхняя граница конкретного stack, но не честное A/B между Codex и OpenCode.

Смена модели обычно влияет сильнее scaffold, но не всегда: в исследовании Gemini 2.5 Pro прибавил 17 percentage points при переходе с OpenHands на Terminus 2.[24] Следовательно, тезис «harness неважен, выбирайте только модель» также неверен. Нужно сначала сравнивать одинаковую model в разных harnesses, затем — лучшие native stacks.

Работа *AI Agents That Matter* предлагает оценивать Pareto frontier по accuracy и cost и критикует сложные agents без адекватных holdouts и воспроизводимости.[25] Это прямо относится к marketing leaderboards: высокий score, полученный большим числом rollouts и tokens, может быть экономически хуже более скромного результата.

### Реальные pull requests не дают единственного победителя

Анализ 7 156 pull requests пяти agents показал разные сильные стороны по типам задач и не нашёл одного лидера для всего: acceptance зависел от класса работы, repository и agent.[35] Такие данные ближе к реальному миру, чем synthetic benchmark, но остаются observational: agents выбирали разные пользователи для разных задач, поэтому причинно приписать разницу одному harness нельзя.

## Производительность разработчика: исследования противоречат только на первый взгляд

Ранний controlled experiment с GitHub Copilot показал, что участники быстрее реализовали короткий JavaScript HTTP server — примерно на 55,8%.[28] Три более крупных field experiments в Microsoft, Accenture и Fortune 100, опубликованные в *Management Science* в 2026 году, охватили 4 867 разработчиков и дали pooled estimate около 26% дополнительных completed tasks; менее опытные разработчики чаще принимали инструмент и получали больший эффект.[27]

Эти результаты поддерживают AI assistance, но в основном измеряют code completion и короткий feedback loop. Они не доказывают, что автономный agent быстрее решает неоднозначные issues в знакомом зрелом repository.

METR провела RCT с 16 опытными open-source developers и 246 реальными задачами в repositories, где участники работали в среднем около пяти лет. При доступе к early-2025 AI tools они тратили на задачи на 19% больше времени, хотя до исследования ожидали ускорение на 24%, а после по-прежнему считали, что ускорились примерно на 20%.[26] Интервал и малая выборка требуют осторожности; исследование не означает «AI всегда замедляет». Оно показывает, что subjective speed не равна measured cycle time и что review/prompt overhead способен съесть выгоду генерации.

Systematic review human-AI software development находит преимущества в code search, repetitive work и скорости, но также cognitive offloading, flow disruption и нестабильное качество. Большинство первичных исследований exploratory или laboratory; longitudinal и team-level evidence мало.[29] Поэтому количество принятых строк, satisfaction или число agent sessions нельзя использовать как самостоятельный ROI.

Практический синтез:

- AI чаще помогает на boilerplate, тестах, документации и unfamiliar APIs;
- сильный maintainer на знакомом codebase может потратить больше времени на объяснение и review;
- cloud delegation может увеличить throughput команды, не уменьшая cycle time отдельной задачи;
- качество workflow и task selection важнее заявления «мы внедрили agent».

## Безопасность и privacy: open source не равно local и не равно safe

Coding agent читает недоверенный repository, issues, web pages и tool output, затем имеет доступ к filesystem и shell. Это классическая граница indirect prompt injection. В InjecAgent ReAct agent на GPT-4 оказался уязвим примерно в 24% из 1 054 сценариев; benchmark общий, но его модель угроз напрямую переносится на README, comments и MCP responses.[32]

Исследования code generation показывают другую проблему: функционально правдоподобный код может быть небезопасен. В IEEE S&P study около 40% программ Copilot в специально подобранных CWE-сценариях содержали уязвимости; это старая model и adversarial task set, поэтому число нельзя считать текущей defect rate.[30] В ACM CCS user study участники с Codex-based assistant писали менее безопасные решения и были увереннее в их безопасности.[31] Оба результата требуют security-aware review, а не запрета на agents.

Сравнивать privacy нужно по отдельным слоям:

1. **Execution:** где выполняются commands и какие paths доступны.

2. **Inference:** local model или remote provider.
3. **Transport:** идёт ли traffic напрямую provider или через intermediary.
4. **Retention/training:** какие policies применяются к plan и organization.
5. **Extensions:** какой код выполняют plugins, MCP servers и skills.
6. **Secrets/network:** может ли agent читать credentials и отправлять данные наружу.

NIST AI 600-1 рекомендует lifecycle risk management, incident response и отдельную оценку third-party GAI risks.[33] Из этого следует конкретная policy: sandbox и network allowlist включаются до первого pilot; secrets выдаются минимально; repository content считается untrusted; destructive commands и external writes требуют approval; каждый patch проходит tests и human review.

Codex даёт наиболее готовую базовую реализацию этой policy. OpenCode позволяет fine-grained permissions. Pi требует container или собственный extension. MiMo Code наследует широкие возможности OpenCode, но из-за возраста продукта нуждается в отдельном audit. Local inference уменьшает exposure model provider, но не защищает от malicious repository и ошибочного shell command.

## Стоимость: считать нужно принятые изменения

Цена подписки удобна для бюджета, но плохо предсказывает стоимость agentic development. Один успешный task включает inference, sandbox/CI compute, неудачные rollouts, ожидание developer, review, повторные исправления и риск rollback. Terminal-Bench показывает попытки стоимостью от единиц до десятков долларов, а отдельные задачи потребляли огромное число API calls и tokens без монотонной связи между расходом и успехом.[24]

Полезная метрика:

`TCO per accepted change = license/API + compute + failed runs + developer time + review/security time + rework + incidents.`

Рядом следует измерять:

- долю задач, принятых без ручного переписывания;
- median и p95 cycle time;
- first-pass CI rate;
- defects и security findings после merge;
- rework через 7 и 30 дней;
- token/cost p50 и p95;
- время автора и reviewer.

Codex может оказаться дешевле OpenCode с тем же OpenAI API благодаря лучшей model/harness integration и caching. OpenCode может оказаться дешевле, если позволяет маршрутизировать простые задачи на Luna, open-weight или local model. Pi может выиграть в узкой автоматизации, но проиграть стоимостью собственной platform engineering. MiMo Code может сократить потери context в длинной задаче, но добавить orchestration overhead. Эти утверждения нужно измерять, а не принимать из architecture description.

## Как провести честный bake-off

Для индивидуального разработчика достаточно 10–15 типичных задач. Для команды нужен набор 30–50 свежих private tasks, разделённых на boilerplate, bug fix, refactor, tests, docs и security. Public benchmark не заменяет этот набор.[25]

### Этап 1: сравнить harness

Запустить одну OpenAI model в Codex, OpenCode и Pi, насколько это допускают продукты. Зафиксировать:

- один prompt и одинаковые repository instructions;
- timeout, reasoning effort и token budget;
- network policy и approvals;
- одинаковый commit/branch baseline;
- минимум три repeats для части задач;
- blinded review результата.

Этот этап отвечает на вопрос, какой harness лучше использует OpenAI model.

### Этап 2: сравнить лучшие stacks

Затем сравнить native Codex + актуальная OpenAI model, OpenCode + лучшая доступная модель, Claude Code + Claude и, при интересе, MiMo Code + MiMo. Здесь одинаковая модель уже не обязательна: вопрос меняется на «какой продукт даёт лучший результат для нашей команды».

### Этап 3: проверить эксплуатацию

Неделя pilot должна включать не только coding:

- восстановление после неудачного compaction;
- работу с большим repository;
- конфликтующие инструкции;
- malicious text в README или issue;
- исчерпание limit и рост стоимости;
- параллельные worktrees;
- audit log и удаление secrets;
- rollback неудачного patch.

Победителем становится не agent с самым красивым demo, а stack с лучшим отношением accepted-task rate к TCO и приемлемым security profile.

## Дискуссионные вопросы и противоречия

### Native integration против provider neutrality

Codex, вероятно, быстрее адаптируется к новым OpenAI models и снижает integration risk. OpenCode и Pi снижают strategic lock-in. Нельзя получить оба преимущества полностью: универсальный abstraction неизбежно сглаживает provider-specific behavior, а first-party harness ограничивает свободу замены provider.

### Богатый harness против прозрачного context

MiMo Code и OpenCode предоставляют много готовых mechanisms; Pi сознательно оставляет primitives. Богатый harness быстрее даёт результат, но добавляет скрытые prompts, compaction и orchestration. Минимальный harness наблюдаем, но требует platform work. Выбор зависит от того, кто будет владеть этой сложностью: vendor или команда.

### Автономность против контроля

Cloud agents и subagents увеличивают throughput, но создают больше действий, которые reviewer не наблюдает в реальном времени. Sandbox уменьшает blast radius, но не доказывает correctness.

Высокая автономность оправдана для well-specified, testable tasks; неоднозначные architecture и security changes требуют коротких checkpoints.

### Производительность против ощущения производительности

Enterprise RCT показывают положительный average effect, а METR — slowdown для опытных maintainers на знакомом codebase.[26][27] Эти результаты совместимы: разные задачи, пользователи и interfaces дают разные эффекты. Поэтому организация должна измерять cycle time и rework, а не опрашивать разработчиков только об ощущении скорости.

## Недостаточность данных

- Нет независимого, воспроизводимого сравнения Codex, OpenCode, Pi и MiMo Code на одной актуальной OpenAI model с одинаковыми prompts, budgets и permissions.
- Vendor comparison MiMo Code с Claude Code использует одинаковую MiMo model, но не публикует достаточный replication package.[14]
- Нет долгосрочных данных о maintainability кода после месяцев использования современных autonomous agents; существующие human studies в основном измеряют completion, короткие tasks или ранние инструменты.[26][27][29]
- Публичные цены и limits меняются быстрее инженерных workflows. Числа в обзоре — срез на 13 июля 2026 года, а не долговременная гарантия.[4][5]
- Зрелость MiMo Code нельзя надёжно оценить через месяц после выпуска. Stars и release velocity не заменяют incident history, enterprise audit и стабильность upgrade path.[15]

## Заключение

Если цель — **эффективно работать именно с OpenAI models**, выбирайте **Codex**. First-party поддержка актуальных models, единая линия local/IDE/desktop/cloud и готовые security controls дают лучший баланс качества, времени настройки и эксплуатационного риска. Используйте local или worktree mode для интерактивных изменений, cloud — для хорошо определённых долгих задач, `workspace-write` и approvals — как default, network — по allowlist, а diff/tests review — всегда.[1][2][3]

Если provider lock-in неприемлем, выбирайте **OpenCode**. Он даёт наиболее полный открытый набор функций без необходимости сначала строить собственный agent platform. Сохраняйте один и тот же task protocol для OpenAI и альтернативных models.

Выбирайте **Pi**, когда вам нужен не готовый продукт, а управляемый agent runtime: прозрачный context, extensions, SDK и RPC. Без container или собственного permission layer использовать его на машине с ценными secrets не следует.[12]

Рассматривайте **MiMo Code** как перспективный pilot для persistent memory и long-horizon orchestration. Для OpenAI-first workflow у него пока нет доказанного преимущества над Codex или OpenCode, а опубликованные benchmark claims требуют независимого повторения.[14][15]

Наконец, не стандартизируйте agent только по leaderboard. Проведите private bake-off, посчитайте TCO per accepted change и сохраните возможность заменить harness. В 2026 году лучший устойчивый выбор — не один вечный инструмент, а **Codex как default плюс открытый portability path через OpenCode**.

## Quality Metrics

| Метрика | Значение |
|---|---|
| Режим | deep |
| Источников найдено | 58 |
| Источников процитировано | 35 |
| Academic | 12 |
| Official / documentation | 20 |
| Industry | 2 |
| Blog | 1 |
| Покрытие проверяемых утверждений цитатами | ~95% |
| Проверены контраргументы | да |
| Sub-questions | 8 |
| Раундов исследования | 2 |
| Вопросов возникло при анализе | 6 |
| Вопросов разрешено | 5 |
| Недостаточно данных | 1 составной вопрос: независимый equal-model benchmark всех четырёх harnesses |

## Примечания


[1] OpenAI. “Codex CLI.” Проверено 2026-07-13. https://developers.openai.com/codex/cli

[2] OpenAI. “Agent approvals & security.” Проверено 2026-07-13. https://learn.chatgpt.com/docs/agent-approvals-security

[3] OpenAI. “Codex cloud.” Проверено 2026-07-13. https://developers.openai.com/codex/cloud

[4] OpenAI. “Introducing GPT-5.6.” 2026-07-09. https://openai.com/index/gpt-5-6/

[5] OpenAI Help Center. “Codex rate card.” Проверено 2026-07-13. https://help.openai.com/en/articles/20001106-codex-rate-card-2

[6] OpenAI. “Enterprise privacy” и “How your data is used to improve model performance.” Проверено 2026-07-13. https://openai.com/enterprise-privacy/ https://help.openai.com/en/articles/5722486-api-data-usage-policies

[7] OpenAI. “Addendum to GPT-5.2 System Card: GPT-5.2-Codex.” 2025-12-18. https://cdn.openai.com/pdf/ac7c37ae-7f4c-4442-b741-2eabdeaf77e0/oai_5_2_Codex.pdf

[8] Anomaly. “OpenCode: The open source coding agent.” Repository, MIT. Проверено 2026-07-13. https://github.com/anomalyco/opencode

[9] OpenCode. “Providers.” Проверено 2026-07-13. https://opencode.ai/docs/providers/

[10] OpenCode. “Permissions.” Проверено 2026-07-13. https://opencode.ai/docs/permissions/

[11] OpenCode. “Enterprise.” Проверено 2026-07-13. https://opencode.ai/docs/enterprise/

[12] Earendil Works. “Pi coding agent.” Repository and README, MIT. Проверено 2026-07-13. https://github.com/earendil-works/pi/tree/main/packages/coding-agent

[13] Mario Zechner. “What I learned building an opinionated and minimal coding agent.” 2025-11-30. https://mariozechner.at/posts/2025-11-30-pi-coding-agent/

[14] Xiaomi MiMo. “MiMo Code Released and Open-Sourced.” 2026-06-15. https://mimo.mi.com/docs/en-US/news/latest/mimocode

[15] XiaomiMiMo. “MiMo Code: Where Models and Agents Co-Evolve.” Repository and Use Restrictions. Проверено 2026-07-13. https://github.com/XiaomiMiMo/MiMo-Code https://github.com/XiaomiMiMo/MiMo-Code/blob/main/USE_RESTRICTIONS.md

[16] Anthropic. “Claude Code security” и pricing. Проверено 2026-07-13. https://code.claude.com/docs/en/security https://claude.com/pricing

[17] Cursor. “Security”, “Data use” и pricing. Проверено 2026-07-13. https://www.cursor.com/security https://cursor.com/data-use https://cursor.com/pricing

[18] Cline. Repository, Apache-2.0, и pricing. Проверено 2026-07-13. https://github.com/cline/cline https://cline.bot/pricing

[19] Aider. Repository, repo map и architect mode. Проверено 2026-07-13. https://github.com/Aider-AI/aider https://aider.chat/docs/repomap.html https://aider.chat/docs/usage/modes.html

[20] GitHub. “Risks and mitigations for the Copilot coding agent” и plans. Проверено 2026-07-13. https://docs.github.com/en/copilot/concepts/agents/cloud-agent/risks-and-mitigations https://github.com/features/copilot/plans

[21] Jimenez, C. E. et al. “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?” ICLR 2024. https://arxiv.org/abs/2310.06770

[22] Zhang, L. et al. “SWE-bench Goes Live!” 2025-05-29. https://arxiv.org/abs/2505.23419

[23] Badertdinov, I. et al. “SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents.” NeurIPS 2025. https://arxiv.org/abs/2505.20411

[24] Merrill, M. A. et al. “Terminal-Bench 2.0: Advancing Agentic Terminal Intelligence with Greater Depth and Diversity.” ICLR 2026. https://arxiv.org/abs/2601.11868

[25] Kapoor, S. et al. “AI Agents That Matter.” 2024-07-01. https://arxiv.org/abs/2407.01502

[26] Becker, J. et al. “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.” METR, 2025-07-10. https://metr.org/Early_2025_AI_Experienced_OS_Devs_Study-paper.pdf

[27] Cui, Z. et al. “The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers.” *Management Science*, 2026-02-27. https://pubsonline.informs.org/doi/10.1287/mnsc.2025.00535

[28] Peng, S. et al. “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.” 2023-02-13. https://arxiv.org/abs/2302.06590

[29] Mohamed, Y. et al. “Human-AI Collaboration in Software Engineering: A Systematic Review.” 2025, обновлено 2026. https://arxiv.org/abs/2507.03156

[30] Pearce, H. et al. “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions.” IEEE S&P 2022. https://arxiv.org/abs/2108.09293

[31] Perry, N. et al. “Do Users Write More Insecure Code with AI Assistants?” ACM CCS 2023. https://arxiv.org/abs/2211.03622

[32] Zhan, Q. et al. “InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents.” Findings of ACL 2024. https://aclanthology.org/2024.findings-acl.624/

[33] NIST. “Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile.” NIST AI 600-1, 2024-07-26. https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence

[34] OpenAI. “Separating signal from noise in coding evaluations.” 2026-07-08. https://openai.com/index/separating-signal-from-noise-coding-evaluations/

[35] Williams, D. J. et al. “Comparing AI Coding Agents: A Task-Stratified Analysis of Pull Request Acceptance.” 2026. https://arxiv.org/abs/2602.08915