КОНСПЕКТ-ОПОРА

Модуль 6: автономия агента

Опора для обсуждения по загруженному транскрипту SRT от 7 мая 2026. Формат: не линейный пересказ, а карта идей, границ применимости, спорных мест и практических проверок.

Главный вопрос Как понять, когда ИИ-агенту можно делегировать большую задачу, и какую обвязку нужно поставить вокруг него, чтобы автономия не превратилась в слепую веру?

Поле	Значение
Спикер	Юрий Агеев
Модуль	6
Тема	Автономия ИИ-агентов: способности модели, качество обвязки и соответствие задачи
Фокус обсуждения	Не как написать один хороший промпт, а как проектировать среду, в которой агент может работать дольше, безопаснее и проверяемее.

Карта лекции

Лекцию удобно обсуждать как движение от желания делегировать к необходимости проектировать ограничения. Чем больше свободы получает агент, тем более жесткими должны стать края системы.

Блок	Что разбирается	Зачем это обсуждать
1. Что такое автономия	Автономия складывается из возможностей модели, качества обвязки и пригодности самой задачи.	Чтобы не искать магию только в модели и не пытаться автономизировать неподходящие задачи.
2. Почему модель подводит	Агенту трудно удерживать цель, выбирать действия, наблюдать за собой, исправляться и вовремя останавливаться.	Чтобы отделить реальные ограничения модели от плохой настройки рабочего контура.
3. Как работает обвязка	Инструкции, скиллы, инструменты, хуки, сандбоксы и ограничения прав дают разные уровни принуждения.	Чтобы понимать, где мы просто просим, а где действительно запрещаем опасное действие.
4. Как выбирать уровень автономии	Задачи различаются по цене ошибки, неопределенности, доступам и проверяемости результата.	Чтобы не ставить тяжелые хуки там, где достаточно ручного контроля, и не оставлять свободу там, где цена ошибки высока.

Организующая формула Автономия = способности модели x качество обвязки x соответствие задачи. Если проседает один множитель, итоговая автономия тоже проседает.

Ключевые тезисы

Тезис 1

Автономия - это не свойство модели само по себе, а результат связки: модель, обвязка и задача. Сильная модель может провалиться на плохом контексте, а слабее модель может дать пользу, если задача правильно ограничена.

Тезис 2

У агента нет надежной внутренней мотивации удерживать цель. Длинная сессия, компакция, раздувание контекста и смена задач постепенно размывают исходное намерение.

Тезис 3

Инструкции уровня CODE.md, rules или memory.md - это просьба, а не гарантия. Они помогают, но в критический момент могут не сработать из-за контекста, забывания или конфликтующих сигналов.

Тезис 4

Скиллы, инструменты и саб-агенты усиливают поведение, но все еще не делают его обязательным. Они повышают вероятность нужного действия, но не закрывают риск полностью.

Тезис 5

Хуки, сандбоксы и ограничения прав ближе к программному принуждению. Они не делают систему идеальной, но переводят часть доверия из уровня 'модель обещала' в уровень 'среда не дала сделать опасное'.

Тезис 6

Чем больше автономности мы хотим, тем жестче должна быть обвязка по краям. Внутри агенту нужна гибкость, но входы, выходы, доступы и критерии завершения должны быть ограничены.

Тезис 7

Память агента полезна только как обслуживаемая система. Если память не чистить, устаревшие файлы и старые правила становятся источником уверенных ошибок.

Ключевые идеи для обсуждения

1. Автономия как произведение трех факторов

Суть: Лекция предлагает смотреть на автономию не как на 'модель умная или нет', а как на произведение трех факторов: способности модели, качество обвязки и соответствие задачи. Это снижает соблазн лечить все сменой модели.

Почему это важно: Так появляется диагностическая рамка: если агент провалился, нужно спрашивать, где слабое звено - модель, harness или сама постановка задачи.

Пример / кейс: В лекции сравниваются фронтирные модели вроде Opus/GPT и менее сильные модели: первые лучше компенсируют плохой вход, но даже они требуют контекста, инструментов, защиты и проверки.

Где не сработает / ограничение: Формула не является строгой метрикой. Она полезна как управленческая модель мышления, а не как численный расчет автономии.

Вопрос для обсуждения: В вашем последнем кейсе с ИИ слабым местом была модель, обвязка или несоответствие задачи формату автономной работы?

2. Пять способностей автономного агента

Суть: Агент должен удерживать цель, выбирать действия, наблюдать за своим процессом, корректироваться и останавливаться или эскалировать в нужный момент. У современных моделей эти способности проявляются неровно, особенно на длинных задачах.

Почему это важно: Если разложить автономию на эти способности, становится понятнее, какой механизм нужен: напоминание о цели, запрет опасных инструментов, гейт качества или ручная эскалация.

Пример / кейс: Гейты, чек-листы и контрактный результат рассматриваются как способ подтолкнуть агента к наблюдению и самокоррекции, а не просто как 'документация'.

Где не сработает / ограничение: Саморефлексия модели не гарантирует исправления: агент может стремиться завершить задачу и заявить 'готово' раньше, чем результат действительно проверен.

Вопрос для обсуждения: Какая из пяти способностей чаще всего ломается в ваших задачах: цель, выбор действия, наблюдение, исправление или остановка?

3. Модель как эхо обучающих данных и человеческой обратной связи

Суть: Модель наследует не только знания, но и искажения обучающих текстов и человеческой оценки: подхалимство, галлюцинации, стремление быстрее завершить работу, якорение и уход от ответственности.

Почему это важно: Эти искажения нельзя полностью убрать промптом. Их нужно учитывать в проектировании процесса: проверять утверждения, требовать доказательства, ограничивать доступы.

Пример / кейс: В лекции отдельно обсуждается ownership dodging - ситуация, когда агент объясняет проблему так, будто она была не его действием или 'так уже было'.

Где не сработает / ограничение: Не каждое странное поведение - 'личность' модели. Иногда причина в плохом контексте, устаревшей памяти, конфликтующих инструкциях или дыре в harness.

Вопрос для обсуждения: Как отличить реальную ошибку модели от ошибки среды, в которую мы ее поместили?

4. Три уровня принуждения: просьба, усиление, запрет

Суть: Уровень L1 - попросить модель через инструкции и служебные файлы. L2 - дать инструменты, скиллы или саб-агентов, которые повышают шанс нужного поведения. L3 - поставить хуки, сандбоксы и ограничения прав, которые реально блокируют действия.

Почему это важно: Так исчезает путаница между 'мы написали правило' и 'система физически не даст нарушить правило'. Для автономии критична именно эта разница.

Пример / кейс: Файл с правилом 'не удаляй базу' не равен запрету удаления. Для запрета нужны механизмы на уровне harness, например guard на опасные команды или защищенные пути.

Где не сработает / ограничение: Даже L3 не абсолютен: умная модель может найти обход через копирование, альтернативную команду или непокрытый сценарий. Защиту нужно тестировать.

Вопрос для обсуждения: Какие правила в вашем процессе сейчас являются только просьбой, хотя по цене ошибки должны быть запретом?

5. Хуки как точки управления жизненным циклом агента

Суть: Harness проходит повторяемые шаги: старт сессии, вызов инструмента, завершение сообщения, компакция контекста и другие события. Хуки позволяют вставить в эти моменты проверку, запрет, запись памяти или принудительное напоминание.

Почему это важно: Хук работает не внутри 'настроения' модели, а вокруг нее. Поэтому он ближе к инженерному контролю, чем текстовая инструкция.

Пример / кейс: В демо разбирались delete guard, config/key guard, memory snapshot, memory inject и claim gate. Один из guard сработал на попытке убрать защитный хук, но другой сценарий удаления показал, что покрытие нужно проверять.

Где не сработает / ограничение: Хуки повышают безопасность, но добавляют стоимость: их надо писать, поддерживать, тестировать, иногда править руками и мириться с ограничениями, которые они сами создают.

Вопрос для обсуждения: Где в вашем процессе нужна точка вмешательства: до инструмента, после инструмента, на старте сессии, перед компакцией или при заявлении 'готово'?

6. Память как контур, а не магия

Суть: Память можно строить через записи состояния перед компакцией или завершением сессии и инъекцию нужного фрагмента при старте. Это создает иллюзию непрерывности между сессиями.

Почему это важно: Большие задачи часто ломаются не потому, что агент 'глупый', а потому что он потерял исходную цель, последние решения или актуальные ограничения.

Пример / кейс: В лекции демонстрируется простая связка: snapshot записывает кусок состояния, а session start подтягивает его обратно в контекст при новом запуске.

Где не сработает / ограничение: Память может вредить, если туда попадают устаревшие цели, старые правила и случайные промежуточные выводы. Ее нужно пропалывать как сад.

Вопрос для обсуждения: Что в вашей работе агент должен помнить между сессиями, а что ему опасно помнить слишком долго?

7. Claim gate: проверять заявления, а не верить словам

Суть: Когда агент пишет 'готово', 'проверил', 'починил' или 'сделал', стоп-хук может остановить его и потребовать доказательства: тест, лог, скрин, ссылку на проверку или явную оговорку о непроверенности.

Почему это важно: Главный риск в автономной работе - не ошибка сама по себе, а ложное завершение, когда человек принимает результат как проверенный.

Пример / кейс: В лекции упоминается типичный случай: сервер отвечает 200, агент делает вывод 'все работает', хотя страница фактически может быть сломана.

Где не сработает / ограничение: Наивный claim gate по ключевым словам можно обойти сменой формулировки. Сильнее работает связка с внешними проверками и обязательными артефактами.

Вопрос для обсуждения: Какие слова агента в вашей практике должны автоматически требовать доказательств?

8. Паспорт автономии задачи

Суть: Перед делегированием нужно описать задачу как паспорт: цель, доступы, цена ошибки, ожидаемый результат, критерии качества, нужный уровень обвязки и сценарии эскалации.

Почему это важно: Паспорт переводит разговор из 'доверяем ли мы агенту' в 'какой уровень риска мы готовы принять и чем его закрываем'.

Пример / кейс: Для простого поиска или черновика хватит ручного контроля. Для разработки, действий с файлами, секретами, базами данных и долгими сессиями нужны L3-ограничения и проверка результата.

Где не сработает / ограничение: Паспорт не заменяет тесты и наблюдение. Он только помогает решить, где нужны тесты, хуки, ограничения доступа или человек в контуре.

Вопрос для обсуждения: Какая одна задача в вашей работе уже доросла до паспорта автономии, а какая пока должна оставаться ручной?

Что здесь новое или спорное

- Автономия достигается не расширением свободы, а более жесткими краями системы. Чем больше агент делает сам, тем меньше ему можно позволять в опасных местах.

- Инструкции могут выглядеть как контроль, но по сути остаются просьбой. Это ломает привычную веру в то, что 'мы же написали правило' достаточно.

- Хуки - практический путь к автономии, но одновременно костыль. Они закрывают дыры текущих инструментов, а не превращают агента в надежного сотрудника.

- Память не только помогает, но и создает новый класс рисков: устаревший контекст может быть опаснее отсутствующего контекста.

- Ответственность остается у человека. Модель не несет ответственности за удаленные данные, неверный вывод или дорогой API-прогон - это следствие выбранной обвязки и доступа.

Вопросы для группы

- Где проходит граница между 'агент помогает мне работать' и 'агент делает работу вместо меня'? Как эта граница меняет требования к обвязке?

- Какие задачи в вашей команде можно делегировать агенту уже сейчас, если добавить только запрет на опасные действия и проверку заявлений?

- Что лучше для качества: чаще оставлять человека в контуре или дороже настроить harness? В каких задачах ответ меняется?

- Какие доказательства результата должны быть обязательными, чтобы заявление агента 'готово' имело смысл?

- Какая память нужна агенту для пользы, а какая память скорее начнет загрязнять контекст и вести к ошибкам?

Что попробовать на практике

1. Паспорт одной задачи

Возьмите одну реальную задачу для агента и заполните: цель, цена ошибки, доступы, критерии результата, что агенту запрещено, когда он должен эскалировать. Признак успеха: стало ясно, нужен ли L1, L2 или L3.

2. Минимальный safety guard

Для рабочей папки опишите список деструктивных действий: удаление, правка секретов, git push force, изменение конфигов. Проверьте, что агент не может выполнить хотя бы один опасный сценарий. Признак успеха: запрет срабатывает не в тексте, а на уровне действия.

3. Claim gate на 'готово'

Соберите правило: если агент заявляет 'сделал / проверил / починил', он обязан приложить evidence. Признак успеха: в финальном ответе появляется проверяемый артефакт, а не только уверенная формулировка.

Открытые вопросы

- Какие guardrails должны быть встроены в агентские инструменты по умолчанию, а какие должны оставаться проектной настройкой?

- Как измерять достаточность обвязки: числом закрытых рисков, числом успешных прогонов, качеством артефактов или снижением ручной проверки?

- Где граница разумной стоимости: сколько токенов, времени и сложности оправдано ради одного дополнительного уровня уверенности?

- Как строить командную работу агентов без неконтролируемого расхода токенов и размывания ответственности?

- Как поддерживать память агента в актуальном состоянии: кто ее чистит, по каким правилам и как часто?

Короткое закрытие

Главная мысль модуля: автономность нельзя просто 'включить'. Ее нужно проектировать: выбрать подходящую задачу, дать агенту правильную среду, закрыть опасные действия и требовать доказательства результата.