# 🤖 Создание доверяемых агентных рабочих процессов с помощь...

# 🔬 Создание доверяемых агентных рабочих процессов с помощью DSL

**Ключевые тезисы:**

- ✅ **Механизм имеет значение**: Доверие к результату зависит не только от вывода, но и от процесса его получения.
- 🎯 **Три ключевых требования к процессу**: Легкость понимания (legibility), сохранение целостности при итерациях (fidelity), точное выполнение (faithful execution).
- 🛠️ **DSL как решение**: Специальный язык (Ash PL) обеспечивает прозрачность, контроль и воспроизводимость сложных рабочих процессов.
- ⚖️ **Компромисс: скорость vs. строгость**: Elicit фокусируется на глубине, качестве и надежности, что требует больше времени.

---

## 🎯 Почему механизм имеет значение

Два системы могут выдать идентичный результат, но уровень доверия к ним будет разным в зависимости от внутреннего процесса. Например, вывод от устаревшей модели и от современной системы, использующей критику и переработку, воспринимаются по-разному. **Механизм — это дизайнерский выбор**, который зависит от задачи, домена и пользователя.

## 🎯 Три ключевых требования Elicit к процессу

При разработке исследовательского агента были выделены три основных критерия:

1. **Процесс должен быть понятным (Legible)**. Его должны легко проверять как люди, так и другие агенты (например, для критики).
2. **Итерации должны сохранять целостность (Retain Fidelity)**. Добавление новых шагов или направлений в работу не должно приводить к "дрейфу" от первоначальной цели пользователя.
3. **Процесс должен точно выполняться (Followed Faithfully)**. Система обязана строго следовать утвержденному плану действий.

Эти требования привели к созданию предметно-ориентированного языка (DSL).

## 🛠️ Язык Ash PL: дизайн и особенности

*Ash PL* — это DSL для агентных рабочих процессов в Elicit, ориентированный на научные исследования и принятие решений.

**Отличительные черты:**

- **Неполный по Тьюрингу**: Нет циклов, рекурсии, мутации. Это чисто функциональный реактивный язык.
- **Ограниченное подмножество Python**: Убраны ненужные возможности, добавлены доменно-специфичные примитивы (например, поиск академических статей, клинических trials).
- **Статическая типизация**: Позволяет быстро находить ошибки и перерабатывать код.

**Пример программы на Ash PL** может описывать процесс конкурентного анализа: поиск в сети, объединение результатов, обогащение источников.

## ⚙️ Архитектура системы: интеграция Ash PL

Система построена вокруг цикла **"написание → интерпретация → переработка"** кода на Ash PL.

**Ключевые компоненты:**

- **Пользовательский интерфейс (UI)**: Взаимодействие с пользователем.

- **Журнал событий (Event Log)**: Append-only лог для управления состоянием (паттерн Event Sourcing).
- **Куратор (Curator)**: Компонент, который пишет и перерабатывает код на Ash PL. Использует лучшие доступные модели (например, Claude через Pi).
- **Песочница (Sandbox)**: Среда для куратора.
- **Python-сервис**: Интерпретирует (выполняет) код на Ash PL.
- **Шлюз (Gateway)**: Централизованная точка для безопасного взаимодействия с API LLM.
- **Кэширующее хранилище (Content Address Store)**: Ключевой элемент для производительности. Позволяет хэшировать и запоминать результаты выражений, чтобы не пересчитывать их заново при каждой интерпретации всей программы.

## 📽️ Демонстрация: исследовательский ландшафт

**Запрос:** "Картографировать компании и институты, инвестирующие в фундаментальные модели для биологии".

**Процесс, управляемый Ash PL:**

1. Уточнение задачи у пользователя (широкий ландшафт vs. конкретика).
2. Последовательное выполнение шагов, закодированных в Ash PL: поиск академических статей, веб-поиск, фильтрация, обогащение данных.
3. Создание **артефакта** — итоговой таблицы с организациями и их атрибутами (созданные модели, модальности, коллаборации).

**Как обеспечивается доверие:**

- **Просмотр кода Ash PL**: Для каждого артефакта можно посмотреть исполняемый код, который его сгенерировал. Это обеспечивает проверяемость.
- **Визуализация процесса**: Графическое представление шагов, автоматически генерируемое из Ash PL. Помогает быстро оценить логику работы.

- **Итеративное развитие**: Пользователь может на естественном языке добавлять новые слои анализа (например, "сравнить open-source и закрытые стратегии"). Система расширяет исходную программу на Ash PL, переинтерпретируя ее целиком, но благодаря кэшированию ранее вычисленные части не пересчитываются.

## 💡 Выводы и рекомендации

DSL — не универсальное решение, а инструмент, который стоит выбирать, если ваши цели совпадают с описанными требованиями (прозрачность, целостность, надежность).

**Что нужно учесть при реализации подобной системы:**

1. **Создание DSL** (желательно на основе популярного языка).
2. **Эргономика для агента** (обертка для смены моделей и harness).
3. **Обработка прерываний** (возможность добавлять новые запросы без остановки процесса).
4. **Восстановление сессий** (rehydration).
5. **Изоляция credentials** (безопасность).
6. **Управление выводом моделей** (парсинг, обработка).
7. **Управление состоянием** (например, через Event Sourcing).
8. **Инвестиции в оценку (eval)** — критически важно в сложных динамических системах.

**Главный посыл:** Инвестируйте в проектирование **механизма** работы вашей системы. Именно детальный, проверяемый и воспроизводимый процесс создает реальное доверие у пользователей, отличая ваш продукт от простого генератора ответов.

---