🧠 Тестовые вычисления Claude: как токены решают сложные з...

🧠 Тестовые вычисления (Test Time Compute): как Claude использует токены для решения сложных задач

Ключевые тезисы:

- 🔥 **Тестовые вычисления** — это использование дополнительных вычислительных ресурсов (токенов) во время инференса (генерации ответа) для улучшения качества решения.
- 📈 Чем больше токенов модель тратит на «размышления», тем лучше результат на сложных задачах.
- 🎛️ Пользователи могут контролировать этот процесс через **уровни усилия (effort)** и **бюджеты (budgets)**.
- 🤖 **Адаптивное мышление (Adaptive Thinking)** позволяет Claude самостоятельно решать, когда «думать», вызывать инструменты или выдавать текст, что делает его работу более похожей на человеческую.

📊 Как работают тестовые вычисления

Принцип: Аналогично тому, как увеличение вычислительных мощностей при обучении (train time compute) делает модель умнее, увеличение количества токенов при генерации ответа (test time compute) повышает качество её работы.

- **Доказательство на графиках:** Производительность Claude (Opus, Sonnet, Haiku) на внутренних бенчмарках (кодирование, DeepSeek QA, PhD-экзамены) линейно растёт с увеличением количества использованных токенов.

Пример с моделированием движения машин:

- 🟢 **Низкое усилие (Low Effort):** Быстро (~50 сек, ~4600 токенов). Создана простая, но функциональная симуляция. Светофор расположен нелогично.
- 🟡 **Высокое усилие (High Effort):** Вдвое больше времени и токенов. Симуляция детальнее, появились разные типы машин, светофор висит над дорогой, водители «умнее».
- 🔴 **Максимальное усилие (Max Effort):** В 10 раз больше ресурсов. Наиболее детализированная и реалистичная симуляция с корректной физикой и сложным поведением машин.

Вывод: Больше токенов → больше времени на «размышление» → более качественный и сложный результат.

🛠️ Три компонента работы Claude во время инференса

1. 🧠 **Мышление (Thinking):** Внутренний «черновик» Claude, пространство для рассуждений и планирования следующих шагов.
2. 🔧 **Вызов инструментов (Tool Calling):** Интерфейс для взаимодействия с внешним миром (веб-поиск, API, файловая система и т.д.).
3. 📝 **Текст (Text):** Финальный или промежуточный вывод, который видит пользователь.

🎚️ Как пользователи управляют процессом: Усилие (Effort) и бюджеты

Уровни усилия (Effort)

Это шкала от **Low** до **Max**, которая контролирует, сколько времени и токенов Claude потратит на задачу. Ключевой вопрос: **«Пожертвовать ли интеллектом ради скорости?»**.

Эволюция управления мышлением:

1. **Изначальный подход:** Claude сначала долго думает, потом выполняет все действия, потом отвечает (неестественно для человека).
2. **Чередующееся мышление (Interleaved Thinking):** Claude может думать после каждого вызова инструмента (более естественно).
3. ✅ **Адаптивное мышление (Adaptive Thinking):** Claude **сам решает**, когда думать, вызывать инструмент или выдавать текст, в зависимости от задачи. Это **парето-эффективное** улучшение.

Важно: Раньше переключатель «мышления» просто отключал эту способность. Теперь же с Adaptive Thinking мы даём Claude инструмент «подумать», а он решает, когда его использовать (аналогично тому, как мы даём инструмент поиска, но не заставляем им всегда пользоваться).

Бюджеты (Budgets)

Более жёсткие ограничения, например, на максимальное количество токенов или время выполнения задачи (**task budgets** в API).

💡 Лучшие практики по выбору уровня усилия

1. 📊 **Оценивайте производительность на своих задачах:** Создайте сложный тестовый набор (eval) и проверьте, как модель справляется на разных уровнях усилия.
2. 📉 **Учитывайте убывающую предельную отдачу:** На самых высоких уровнях (Extra High → Max) прирост качества может быть незначительным при резком росте затрат.

3. **Правила на подбор уровня (от сложного к простому):**
    - **Max:** Только для самых сложных задач, где критически важен интеллект. Может быть избыточен.
    - **Extra High (рекомендация по умолчанию):** Оптимальный баланс интеллекта, скорости и количества токенов. Используется в Claude Code и Claude.ai.
    - **High:** Хороший баланс. Следует использовать, если задача требует хоть какого-то интеллекта.
    - **Medium/Low:** Для задач, чувствительных к задержкам (латентности), где интеллект не главное: классификация, суммаризация, извлечение данных.
4. ⚠️ **Низкое усилие может давать неожиданные решения:** В тесте «Claude Plays Pokémon» на Low Effort модель использовала хитрые тактики (репелленты, побеги), чтобы пройти игру быстрее, избегая размышлений.

🤔 Haiku vs. Opus: что выбрать?

- **Большая модель (Opus) на низком усилии** часто даёт лучший результат, чем **маленькая модель (Haiku) на высоком усилии**, если задача требует интеллекта.
- **Маленькие модели (Haiku)** лучше подходят для простых, низкоинтеллектуальных задач, где результат предсказуем, а скорость и стоимость важны.
- **Идеальный способ:** Протестировать обе модели на всех уровнях усилия с помощью своих эвалов.

✅ Итоговые рекомендации

1. **Всегда включайте мышление:** Давайте Claude возможность использовать инструмент «подумать» (через Adaptive Thinking).
2. **Контролируйте глубину через Effort:** Используйте уровни усилия, чтобы найти баланс между качеством, скоростью и стоимостью.

3. **Создавайте сложные эвалы:** Лучший способ определить оптимальные настройки — тестирование на самых трудных представительных задачах.
4. **По умолчанию — Extra High:** Если сомневаетесь, начните с этого уровня, как это делает Anthropic в своих продуктах.
5. **Цель на будущее:** Дать пользователям возможность ставить Claude **бюджет** (деньги или время), а модель сама будет оптимально распределять свои вычислительные ресурсы (токены) для его выполнения.
