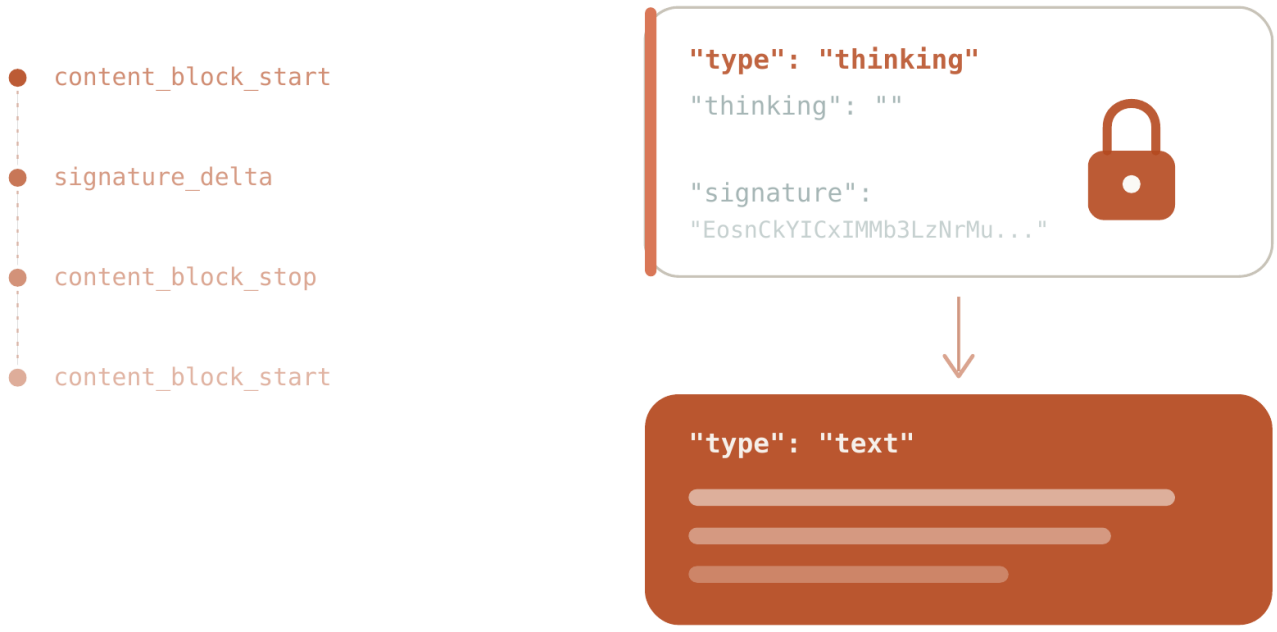
ПОД КАПОТОМ

Мышление Claude

Как устроено adaptive thinking: за что вы платите, что видите в ответе и где мышление обнуляет ваш кэш.



Under the Hood · Илья Утов
@gorilla_under_hood

Содержание

Как читать этот перевод

Anthropic переписала страницу документации про мышление Claude. Это уже не тот «extended thinking», к которому все привыкли: у страницы новое имя, новый параметр и новое поведение по умолчанию. Сам документ слова «legacy» про extended thinking не произносит, но отправляет за его настройкой на отдельную страницу и адресует её моделям, которые не умеют иначе.

Ниже полный перевод всех одиннадцати содержательных разделов. Двенадцатый, «Next steps», это блок ссылок, он разошёлся по разделу «Источники». Три отступления от оригинала сделаны сознательно.

Первое: примеры кода в оригинале даны на десяти языках, здесь оставлен Python и голые JSON. Второе: перекрёстные ссылки на смежные страницы документации собраны в раздел «Источники» в конце. Третье: подзаголовки внутри некоторых разделов расставлены мной для навигации по A4.

Дополнено после публикации. Раздел «Уровни усилий и цена» добавлен по замечанию читателя из комментариев в LinkedIn: на исходной странице этой таблицы нет, а без неё легко решить, что уровень усилий это лимит на токены мышления. Это не так, и разница видна в счёте.

Что моё, а что Anthropic. Основной текст это перевод. Мои комментарии вынесены во врезки с пометкой «Под капотом» и в последний раздел «Разбор от канала».

Главное в одном абзаце. На всех моделях пятого поколения мышление включено по умолчанию, но по умолчанию же не показывается: `display` там стоит в `omitted`, и вы получаете пустой блок с подписью. Токены за это списываются полностью. Пропуск вывода экономит время, но не деньги. Отсюда самая частая ошибка чтения: разработчик видит пустое поле `thinking` и решает, что мышление выключено, хотя оно работало и было оплачено.

Как переведены термины

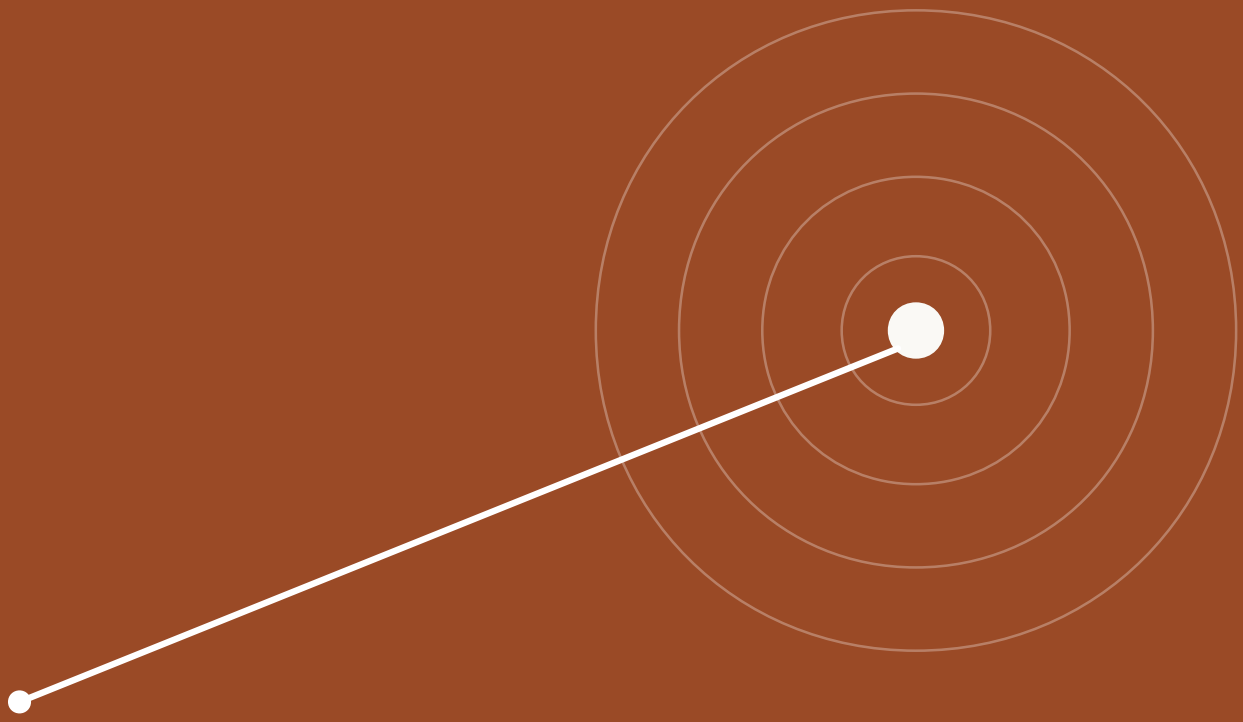
Оригинал	В переводе	Что это
adaptive thinking	адаптивное мышление	Актуальный режим: модель сама решает, думать ли и насколько глубоко
extended thinking	ручной режим	Прежний режим с бюджетом в токенах, для моделей, которые не умеют иначе
thinking block	блок мышления	Элемент ответа с типом `thinking`. Имя типа в коде не переводится
display	показ мышления	Поле, которое решает, вернуть текст сводки или пустое поле. Значение `omitted` = мышление не показано
summarized	сводка	Пересказ рассуждений, который делает отдельная модель
signature	подпись	Зашифрованная копия полных рассуждений внутри блока
redacted_thinking	блок, скрытый по безопасности	Не путать с пустым полем при `display: "omitted"`: это отдельный тип блока
turn	ход	Одно завершённое высказывание участника. Цикл с инструментами это один ход
effort	уровень усилий	Параметр, который решает, сколько работы модель вкладывает в ответ
interleaved thinking	чередующееся мышление	Мышление между вызовами инструментов, а не только перед ответом
harness	обвязка	Ваш каркас вокруг модели: цикл вызовов, инструменты, системные инструкции

ЧАСТЬ · I

I

Как это работает

Что происходит внутри ответа, когда модель думает, и как мышление включается.



Как устроено мышление

Модель без мышления обязана ответить с первой попытки: без черновика, без проверки, без смены курса на середине. Для доказательства, хитрого бага или длинной агентной задачи первый подход часто оказывается не лучшим.

Мышление снимает это ограничение. Claude проходит задачу своими словами до ответа: переформулирует вопрос, пробует подходы, проверяет промежуточные результаты и бросает пути, которые не выдержали проверки. Эти рассуждения приходят в блоках `thinking` перед ответом, и Claude опирается на них, когда строит финальный текст. Поэтому мышление улучшает результат на сложных задачах вроде математики, кода, анализа и долгой агентной работы, где качество ответа зависит от промежуточной работы, которая иначе была бы сжата в сам ответ или пропущена.

У мышления есть цена. Токены, потраченные на рассуждения, **идут в счёт как выходные, даже когда текст мышления вам не возвращается**, и считаются в `max_tokens` вместе с текстом ответа.

Будет ли Claude думать на конкретном запросе и насколько глубоко, зависит от вашей конфигурации мышления и сложности запроса.

Вот как мышление выглядит в ответе: один или несколько блоков `thinking` приходят перед блоками `text`. Блок мышления это такой же сгенерированный контент, как идущий за ним блок `text`, но он отделён от основного ответа. Каждый блок мышления несёт поле `signature`, зашифрованную копию полных рассуждений, которую вы передаёте обратно без изменений в многоходовых диалогах и при работе с инструментами.

ИЗ ДОКУМЕНТАЦИИ

```
{
  "content": [
    {
      "type": "thinking",
      "thinking": "Let me break this down. The question has two parts, so I'll
start with the simpler one and use its result to constrain the second...",
      "signature": "WaUjzkypQ2mUEVM36O2Txu...."
    },
    {
      "type": "text",
      "text": "Based on my analysis..."
    }
  ]
}
```

Этот текст видно не всегда, и то, что вы видите, никогда не бывает полной цепочкой рассуждений без обработки: текст в блоке мышления это **сводка** рассуждений Claude. Поле `display` в конфигурации мышления управляет тем, вернётся ли эта сводка вообще. Значение `"summarized"` её возвращает, а `"omitted"`, значение по умолчанию на новейших моделях, возвращает блоки мышления с пустым полем `thinking`. В обоих случаях блок оплачивается одинаково и одинаково передаётся обратно в многоходовых диалогах.

ПОД КАПОТОМ · САМАЯ ЧАСТАЯ ОШИБКА ЧТЕНИЯ

Пустое поле `thinking` в ответе не означает, что модель не думала. На всех моделях пятого поколения это дефолт: модель думала, токены списаны, текст вам просто не отдали.

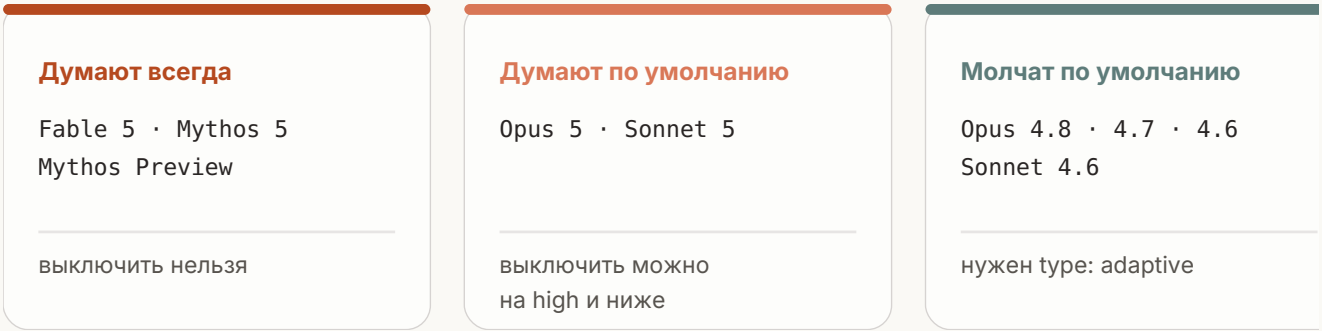
Проверять факт мышления надо по наличию блока, а не по наличию текста в нём.

Если Claude использует инструменты, мышление может появляться и между их вызовами.

Структура ответа с мышлением

Настройка мышления

На актуальных моделях мышление включено по умолчанию или включается одним параметром.



Три режима по моделям

На Claude Opus 5, Claude Sonnet 5, Claude Fable 5, Claude Mythos 5 и Claude Mythos Preview мышление уже включено, настраивать нечего. Первое, что обычно нужно разработчику на этих моделях, это **увидеть** текст мышления, поскольку `display` там по умолчанию стоит в `"omitted"`. Включается через `thinking: {"type": "adaptive", "display": "summarized"}`.

На Claude Opus 4.8, Claude Opus 4.7, Claude Opus 4.6 и Claude Sonnet 4.6 мышление выключено, пока вы не поставите `thinking: {type: "adaptive"}`.

ИЗ ДОКУМЕНТАЦИИ

```
client = anthropic.Anthropic()

response = client.messages.create(
    model="claude-opus-4-8",
    max_tokens=16000,
    thinking={"type": "adaptive", "display": "summarized"},
    messages=[
        {
            "role": "user",
            "content": "What is the greatest common divisor of 1071 and 462?",
        }
    ],
)

for block in response.content:
    if block.type == "thinking":
        print(f"\nThinking: {block.thinking}")
    elif block.type == "text":
        print(f"\nResponse: {block.text}")
```

Запуск примера печатает сводку мышления, а затем ответ:

ИЗ ДОКУМЕНТАЦИИ

```
Thinking: Use Euclidean algorithm.
1071 = 2*462 + 147
462 = 3*147 + 21
147 = 7*21 + 0
GCD = 21

Response: ## Finding GCD of 1071 and 462

I'll use the **Euclidean algorithm**, repeatedly dividing and taking
remainders...
```

Токены мышления считаются в `max_tokens`, поэтому ставьте его достаточно высоким, чтобы хватило и на мышление, и на текст ответа.

Как выключить мышление

На Claude Sonnet 5, где мышление включено по умолчанию, его можно выключить:

ИЗ ДОКУМЕНТАЦИИ

```
client = anthropic.Anthropic()

response = client.messages.create(
    model="claude-sonnet-5",
    max_tokens=4096,
    thinking={"type": "disabled"},
    messages=[{"role": "user", "content": "Summarize this article in one
sentence."}],
)
```

У Claude Opus 5 мышление тоже включено по умолчанию, и он принимает `thinking: {type: "disabled"}` на уровне усилий `high` и ниже. На `xhigh` и `max` мышление выключить нельзя: запросы, которые совмещают `thinking: {type: "disabled"}` с этими уровнями усилий, возвращают ошибку 400. Ограничение действует на Claude Opus 5 и более поздних моделях и проверяется на каждом запросе. С выключенным мышлением Claude Opus 5 может изредка выдавать вызовы инструментов обычным текстом или включать внутренние XML-теги в видимый вывод.

Claude Fable 5, Claude Mythos 5 и Claude Mythos Preview отклоняют `thinking: {type: "disabled"}`: на этих моделях мышление выключить нельзя.

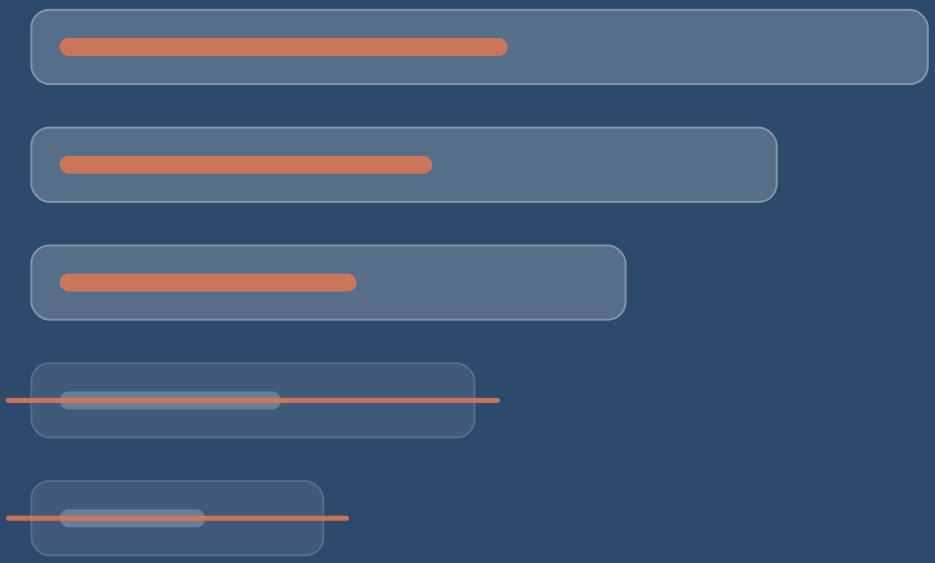
Если ваша модель поддерживает только extended thinking, настраивайте его через `type: "enabled"` и значение `budget_tokens`.

ЧАСТЬ · II

II

Что видно в ответе

Поле display, сводка вместо цепочки рассуждений, стриминг и связь с уровнем усилий.



Что видно в ответе

Поле `display` управляет тем, как контент мышления возвращается в ответах API. Оно работает в обоих режимах: ставьте его рядом с `type: "adaptive"` или `type: "enabled"`. Принимает два значения.

`"summarized"`: блоки мышления содержат текст сводки, читаемое изложение рассуждений Claude. Это значение по умолчанию на Claude Opus 4.6, Claude Sonnet 4.6 и более ранних моделях.

`"omitted"`: блоки мышления возвращаются с пустым полем `thinking`. Поле `signature` при этом по-прежнему несёт зашифрованные полные рассуждения для непрерывности многоходового диалога. Это значение по умолчанию на Claude Fable 5, Claude Mythos 5, Claude Opus 5, Claude Sonnet 5, Claude Opus 4.8, Claude Opus 4.7 и Claude Mythos Preview.

Ставьте `display: "omitted"`, когда ваше приложение не показывает мышление пользователю. Основная выгода это **более быстрый выход первого текстового токена при стриминге**: сервер вообще пропускает стриминг токенов мышления и отдаёт только подпись, поэтому финальный текст начинает стримиться раньше.

ИЗ ДОКУМЕНТАЦИИ

```
{
  "content": [
    {
      "type": "thinking",
      "thinking": "",
      "signature": "EosnCkYICxIMMb3LzNrMu..."
    },
    {
      "type": "text",
      "text": "The answer is 12,231."
    }
  ]
}
```

Что важно помнить, когда мышление не показано:

› Вам всё равно выставляют счёт за полные токены мышления. Так экономится время, но не деньги.

- Если вы передаёте блоки мышления обратно в многоходовых диалогах, передавайте их без изменений. Сервер расшифровывает `signature`, чтобы восстановить исходное мышление при сборке промпта. Любой текст, который вы положите в поле `thinking` такого блока, когда возвращаете его обратно, игнорируется.
- `display` недопустим вместе с `thinking.type: "disabled"`: показывать нечего.
- При `thinking.type: "adaptive"`, если модель пропустила мышление на простом запросе, блок мышления не создаётся вообще, независимо от `display`.
- При стриминге с `display: "omitted"` события `thinking_delta` не отправляются.

Поле `signature` одинаково при обоих значениях `display`. Переключать значения `display` между ходами диалога можно.

	`"summarized"`	`"omitted"`
Текст мышления	сводка рассуждений	пустая строка
Оплата токенов	полная	полная, та же
Стриминг	идут thinking_delta	только signature_delta
Где дефолт	Opus 4.6, Sonnet 4.6 и старше	все модели 5-го поколения

Поле display: что меняется, а что нет

В Ruby SDK это поле задаётся как `display_:` с подчёркиванием на конце, чтобы не перекрыть `Kernel#display`. В самом протоколе поле по-прежнему называется `display`.

Сводка вместо цепочки рассуждений

Когда `display` стоит в `"summarized"`, текст мышления, который вы получаете, это пересказ полного процесса мышления Claude, а не сами рассуждения. Сводка даёт полную интеллектуальную выгоду от мышления, но предотвращает злоупотребления. **Полные рассуждения без обработки не возвращает ни одна настройка `display`.**

Что важно помнить:

- Счёт выставляется за полные токены мышления исходного запроса, а не за токены сводки. Оплаченное число выходных токенов **не совпадает** с числом токенов, которое вы видите в ответе.
- На Claude Opus 4.6, Claude Sonnet 4.6 и более ранних моделях первые несколько строк вывода мышления более многословны и дают подробные рассуждения, что особенно полезно для промпт-инжиниринга. Claude Mythos Preview делает сводку с первого токена, поэтому этой развёрнутой преамбулы в его блоках мышления нет.
- Сводка сохраняет ключевые идеи процесса мышления с минимальной добавленной задержкой, что позволяет стримить её пользователю.
- Сводку делает **другая модель**, не та, к которой вы обращаетесь. Модель, которая думала, свою сводку не видит.
- Поведение сводки может меняться по мере доработки функции.

В редких случаях, когда вам нужен доступ к полному выводу мышления, Anthropic предлагает связаться с отделом продаж.

Стриминг мышления

Мышление работает со стримингом. Блоки мышления приходят как события `thinking_delta` внутри событий `content_block_delta`, за которыми следует единственное событие `signature_delta` прямо перед `content_block_stop` этого блока. Текстовые блоки стримятся после, как обычно.

ИЗ ДОКУМЕНТАЦИИ

```
client = anthropic.Anthropic()

with client.messages.stream(
    model="claude-opus-4-8",
    max_tokens=16000,
    thinking={"type": "adaptive", "display": "summarized"},
    messages=[
        {
            "role": "user",
            "content": "What is the greatest common divisor of 1071 and 462?",
        }
    ],
) as stream:
    for event in stream:
        if event.type == "content_block_start":
            print(f"\nStarting {event.content_block.type} block...")
        elif event.type == "content_block_delta":
            if event.delta.type == "thinking_delta":
                print(event.delta.thinking, end="", flush=True)
            elif event.delta.type == "text_delta":
                print(event.delta.text, end="", flush=True)
```

При стриминге с включённым мышлением вы можете заметить, что текст иногда приходит крупными кусками вперемешку с более мелкой, потокеновой выдачей. Это ожидаемое поведение, особенно для контента мышления. Система стриминга обрабатывает контент пачками для оптимальной производительности, отсюда «кусочная» выдача и возможные задержки между событиями.

Полная трассировка событий

ИЗ ДОКУМЕНТАЦИИ

```
event: message_start
data: {"type": "message_start", "message": {"id": "msg_01...", "type": "message",
"role": "assistant", "content": [], "model": "claude-opus-4-8", "stop_reason":
null, "stop_sequence": null}}

event: content_block_start
data: {"type": "content_block_start", "index": 0, "content_block": {"type":
"thinking", "thinking": "", "signature": ""}}

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type":
"thinking_delta", "thinking": "I need to find the GCD of 1071 and 462 using the
Euclidean algorithm.\n\n1071 = 2 × 462 + 147"}}

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type":
"thinking_delta", "thinking": "\n462 = 3 × 147 + 21\n147 = 7 × 21 + 0\n\nSo
GCD(1071, 462) = 21"}}

// Additional thinking deltas...

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type":
"signature_delta", "signature": "EqQBCgIYAhIM1gbcDa9GJwZA2b..."}}

event: content_block_stop
data: {"type": "content_block_stop", "index": 0}
```

Дальше блок мышления закрыт, и начинается текст ответа:

ИЗ ДОКУМЕНТАЦИИ

```
event: content_block_start
data: {"type": "content_block_start", "index": 1, "content_block": {"type":
"text", "text": ""}}

event: content_block_delta
data: {"type": "content_block_delta", "index": 1, "delta": {"type": "text_delta",
"text": "The greatest common divisor of 1071 and 462 is **21**."}}

// Additional text deltas...

event: content_block_stop
data: {"type": "content_block_stop", "index": 1}

event: message_delta
data: {"type": "message_delta", "delta": {"stop_reason": "end_turn",
"stop_sequence": null}}

event: message_stop
data: {"type": "message_stop"}
```

Когда стоит `display: "omitted"`, блок мышления открывается, приходит единственный `signature_delta`, и блок закрывается без единого события `thinking_delta`. Стриминг текста начинается сразу после:

ИЗ ДОКУМЕНТАЦИИ

```
event: content_block_start
data: {"type":"content_block_start","index":0,"content_block":
{"type":"thinking","thinking":"","signature":""}}

event: content_block_delta
data: {"type":"content_block_delta","index":0,"delta":
{"type":"signature_delta","signature":"EosnCkYICxIMMb3LzNrMu..."}}

event: content_block_stop
data: {"type":"content_block_stop","index":0}

event: content_block_start
data: {"type":"content_block_start","index":1,"content_block":
{"type":"text","text":""}}
```

Мышление и уровень усилий

Параметр `thinking` управляет тем, думает ли Claude в блоках мышления перед ответом. Параметр `effort` управляет тем, сколько работы Claude тратит на весь ответ целиком, а в адаптивном режиме это включает и то, как часто и как глубоко он думает. Не передавайте `adaptive` как значение `effort`: `adaptive` это режим мышления, а не уровень усилий.

На Claude Opus 4.5, единственной модели с одним лишь extended thinking, которая поддерживает усилия, `effort` сочетается с `budget_tokens`.

Когда два рычага разведены таким образом, берите тот, который отвечает вашей цели:

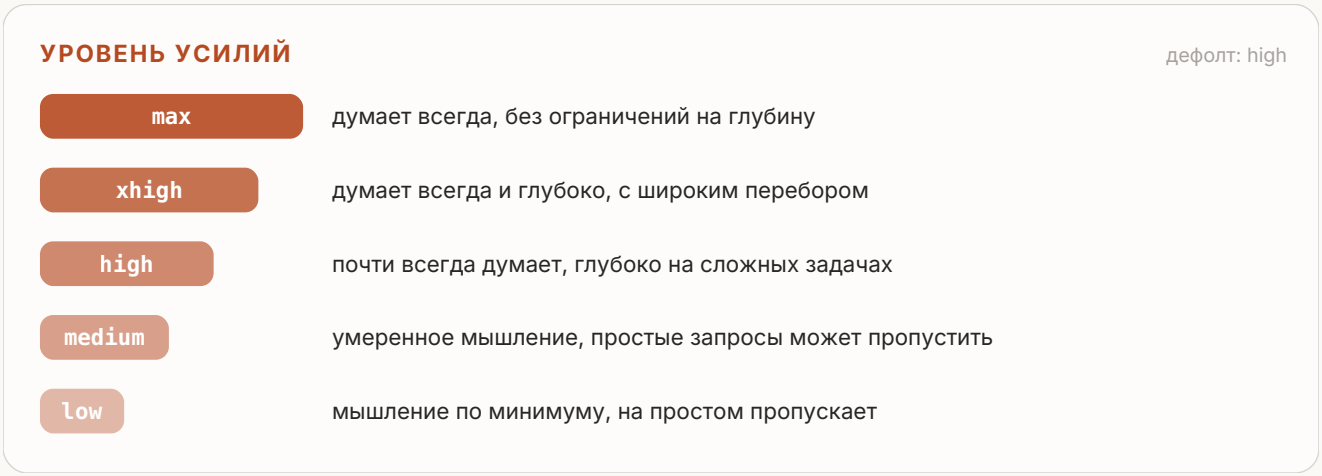
- **Снизить стоимость или задержку на нагрузке с мышлением:** сначала понижайте `effort`. Он масштабирует вниз весь ответ, включая мышление.
- **Claude думает слишком редко или слишком поверхностно:** поднимайте `effort`.
- **Мышление нужно полностью выключить:** используйте `thinking: {type: "disabled"}` на моделях, которые это позволяют.
- **Нужен жёсткий потолок расходов:** используйте `max_tokens`. Усилия это мягкая рекомендация, `max_tokens` это строгий лимит.

ПОД КАПОТОМ · ГДЕ ЗДЕСЬ ЛОВУШКА ДЛЯ АГЕНТА

Понижать `effort` кажется самым дешёвым рычагом экономии. Но конфигурация усилий вкладывается в промпт, поэтому каждое её изменение начинает кэш заново.

Агент, который динамически крутит усилия между шагами, оплачивает промах кэша на каждом переключении. Экономия на мышлении оборачивается переплатой за вход.

Уровни усилий и цена, дополнение



Не лимит токенов, а мягкое указание, как часто и как глубоко думать

Этой таблицы на переведённой странице нет: она отправляет за ней на соседнюю страницу про управление мышлением и стоимость. Собираю сюда, потому что без неё раздел выше повисает в воздухе.

Важное уточнение по механике. Уровень усилий это **не лимит на число токенов мышления**. В документации сказано прямо: усилия это мягкое указание на то, какую долю вывода Claude отдаёт мышлению, оно формирует поведение, но не гарантирует число токенов. Жёсткий потолок один и он называется `max_tokens`, и считает он мышление и ответ вместе.

Отсюда следствие, которое легко упустить. Фраза «усилия управляют тем, сколько модель думает, а не тем, сколько она говорит» верна на уровне API, но в счёте всё иначе: токены мышления тарифицируются как выходные, то есть по самой дорогой ставке, и при этом в видимый ответ не попадают. Практически модель говорит больше, чем вы видите, и платите вы именно за это.

Проверить, сколько ушло на мышление, можно точно. В ответе есть счётчик:

ИЗ ДОКУМЕНТАЦИИ

```
{
  "usage": {
    "input_tokens": 25,
    "output_tokens": 348,
    "output_tokens_details": {
      "thinking_tokens": 312
    }
  }
}
```

В этом примере из 348 выходных токенов 312 ушли на мышление, то есть 90 процентов счёта за ответ. Вычтите `thinking_tokens` из `output_tokens`, чтобы прикинуть объём собственно ответа.

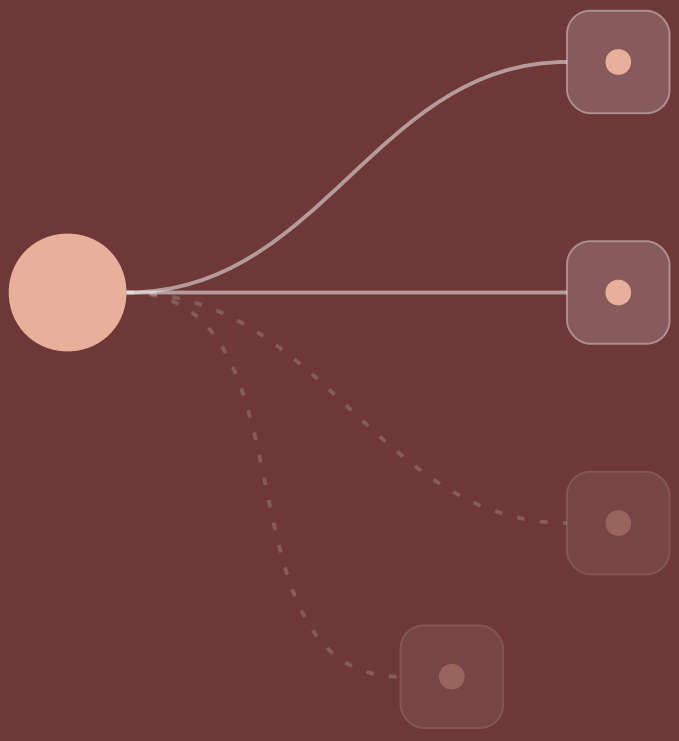
Если запрос упёрся в `stop_reason: "max_tokens"`, документация предлагает два хода: поднять `max_tokens`, чтобы хватило и на мышление, и на ответ, либо понизить уровень усилий, чтобы модель думала меньше и оставила больше бюджета тексту.

ЧАСТЬ · III

III

Мышление и инструменты

Правила хода, сохранение блоков, чередующееся мышление и разница между моделями.



Мышление и инструменты

Мышление работает вместе с инструментами, позволяя Claude рассуждать о выборе инструмента и обрабатывать его результаты. Действуют два ограничения.

Ограничение на выбор инструмента в ручном режиме. Использование инструментов с ручным extended thinking (`thinking: {type: "enabled"}`) поддерживает только `tool_choice: {"type": "auto"}` (значение по умолчанию) или `tool_choice: {"type": "none"}`. Значения `{"type": "any"}` и `{"type": "tool", "name": "..."}` приводят к ошибке, потому что они принуждают к использованию инструмента, что несовместимо с ручным extended thinking. Адаптивное мышление, включая модели, где мышление включено по умолчанию, принудительный вызов инструментов поддерживает.

Сохранение блоков мышления. Когда вы возвращаете результаты инструмента, вы обязаны передать блоки мышления из сообщения ассистента обратно в API, целиком и без изменений.

Цикл работы с инструментами это один ход ассистента. С точки зрения модели ход ассистента не завершается, пока Claude не закончит полный ответ, который может включать несколько вызовов инструментов и их результатов. Вся эта последовательность и есть один ход:

ИЗ ДОКУМЕНТАЦИИ

```
User: "What's the weather in Paris?"
Assistant: [thinking] + [tool_use: get_weather]
User: [tool_result: "20°C, sunny"]
Assistant: [text: "The weather in Paris is 20°C and sunny"]
```

Весь ход идёт в одном режиме мышления: переключить мышление посреди хода нельзя, в том числе внутри цикла работы с инструментами. В ручном режиме API дополнительно требует, чтобы финальный ход ассистента в запросе с мышлением начинался с блока мышления. Адаптивный режим это требование снимает.

Цикл работы с инструментами это один ход

Конфликты посреди хода не роняют запрос. Если вы переключите мышление посреди хода, например между отправкой вызова инструмента и возвратом его результата, API не отдаст ошибку. Вместо этого он молча выключит мышление для этого запроса. Чтобы сохранить качество модели, API может вырезать блоки мышления, которые создали бы недопустимую структуру хода, или выключить мышление, когда история диалога несовместима с включённым мышлением. Чтобы убедиться, что мышление было активно, проверяйте наличие блоков `thinking` в ответе.

Переключайте между ходами, а не внутри них. Планируйте стратегию мышления в начале каждого хода. Завершите ход ассистента, затем меняйте конфигурацию для следующего:

ИЗ ДОКУМЕНТАЦИИ

```
User: "What's the weather?"
Assistant: [tool_use] (thinking disabled)
User: [tool_result]
Assistant: [text: "It's sunny"]
User: "What about tomorrow?"
Assistant: [thinking] + [text: "..."] (thinking enabled - new turn)
```

Учтите, что переключение режимов мышления также сбрасывает кэш промпта.

Как сохранять блоки мышления

Когда Claude вызывает инструмент, он приостанавливает сборку своего ответа в ожидании внешней информации. Когда вы возвращаете результат, Claude продолжает строить тот же самый ответ, поэтому его прежние рассуждения должны быть на месте. Передавайте каждый блок `thinking` обратно в API целиком и без изменений, вместе с блоком `tool_use`, которому он сопутствовал. Это важно по двум причинам.

Непрерывность рассуждений: блоки мышления фиксируют пошаговые рассуждения, которые привели к запросам инструментов. Их наличие позволяет Claude продолжить рассуждать с того места, где он остановился.

Сохранение контекста: результаты инструментов приходят как пользовательские сообщения в структуре API, но принадлежат одному непрерывному потоку рассуждений. Сохранение блоков мышления удерживает этот поток между вызовами API.

Коротко:

- **Обязательно:** внутри хода с инструментами блоки мышления передавать обратно.
- **Рекомендуется:** между ходами передавать обратно всё.
- **Допустимо:** вне работы с инструментами мышление прошлых ходов опускать.

Чистить старое мышление вручную не нужно. Передавайте все блоки обратно, а API отфильтрует их сам, оставит те, что нужны для сохранения рассуждений модели, и выставит счёт за входные токены только за блоки, реально показанные Claude. Какие именно блоки прошлых ходов сохраняются, зависит от модели. Чтобы переопределить поведение по умолчанию, используйте стратегию редактирования контекста `clear_thinking_20251015`.

Внутри последнего сообщения ассистента последовательность идущих подряд блоков `thinking` обязана совпадать с тем, что модель сгенерировала в исходном запросе: их нельзя переставлять, редактировать или выбрасывать частично. Это касается и блоков `redacted_thinking`.

Изменённые блоки мышления отклоняются с ошибкой 400. Единственное исключение: текст, положенный в пустое поле `thinking` непоказанного блока, игнорируется, а не отклоняется.

Чередующееся мышление

Чередующееся мышление позволяет Claude думать между вызовами инструментов, рассуждая о каждом результате до того, как действовать дальше. С ним Claude может рассуждать о результатах вызова перед решением, что делать следующим, сцеплять несколько вызовов с шагами рассуждений между ними и принимать более тонкие решения на основе промежуточных результатов.

Последовательные вызовы инструментов **не требуют** чередующегося мышления. Claude может сцеплять вызовы и с ним, и без него. Чередование меняет то, где появляются блоки мышления между вызовами, а не саму возможность сцеплять вызовы.

При адаптивном мышлении чередование работает автоматически на каждой модели, которая поддерживает адаптивное мышление, бета-заголовок не нужен. На Claude Fable 5, Claude Mythos 5, Claude Mythos Preview, Claude Opus 5, Claude Opus 4.8 и Claude Opus 4.7 рассуждения между вызовами инструментов всегда попадают в блоки мышления. Claude Haiku 4.5 чередующееся мышление не поддерживает. На моделях с ручным extended thinking чередование требует бета-заголовка и меняет способ подсчёта бюджета мышления.

С чередующимся мышлением выделенный объём мышления может распространяться на весь ход ассистента, а не на один ответ. Чередующееся мышление поддерживается только для инструментов, используемых через Messages API.

Сохранение блоков по моделям

Останутся ли блоки мышления из прошлых ходов ассистента в контексте по умолчанию, зависит от модели.

Хранят все прошлые ходы: Claude Opus 4.5 и более поздние Opus, Claude Sonnet 4.6 и более поздние Sonnet, Claude Fable 5, Claude Mythos 5 и Claude Mythos Preview.

Хранят только последний ход: более ранние Opus и Sonnet, а также все Haiku вплоть до Claude Haiku 4.5. Когда вы передаёте более старые блоки обратно, API вырезает их автоматически.

Что происходит с блоками прошлых ходов

Сохранение даёт две выгоды. Оптимизация кэша: сохранённые блоки дают попадания в кэш при работе с инструментами, поскольку передаются обратно вместе с результатами и кэшируются инкрементально в течение хода ассистента, что экономит токены в многошаговых сценариях. И отсутствие влияния на интеллект: сохранение блоков не ухудшает работу модели.

Плата за это идёт контекстом: на моделях, которые хранят всё, длинные диалоги съедают больше места, поскольку сохранённые блоки идут во вход наравне с остальной историей. В обоих режимах поведение автоматическое: ни изменений в коде, ни бета-заголовков не требуется, а блоки мышления надо по-прежнему передавать обратно целиком и без изменений. Чтобы переопределить дефолт в любую сторону, используйте очистку блоков мышления.

Смена модели посреди диалога. Когда вы переключаетесь между любыми двумя моделями, например после отката по отказу классификатора, вырезайте блоки `thinking` и `redacted_thinking` из прошлых ходов ассистента. Блоки мышления привязаны к модели, которая их произвела. Другие модели молча их игнорируют, а не отклоняют запрос, но **проигнорированные блоки всё равно добавляют входные токены**.

ПОД КАПОТОМ · ДВОЙНАЯ ОПЛАТА В ДЛИННОЙ СЕССИИ

На моделях, которые хранят все прошлые ходы, мышление один раз оплачивается как выходные токены, а потом многократно как входные, пока живёт диалог.

Для долгой агентной сессии это означает, что счёт растёт быстрее, чем растёт видимая переписка. Инструмент против этого называется thinking block clearing.

ЧАСТЬ · IV

IV

Кэш и контекст

Где мышление обнуляет кэш, как оно копится в окне и как устроено шифрование блоков.

Мышление и кэш промпта

Кэш промпта взаимодействует с мышлением несколькими конкретными способами. Правила ниже действуют в обоих режимах мышления.

Изменения конфигурации сбрасывают кэш. Конфигурация мышления и итоговый уровень `effort` встраиваются в сам промпт, поэтому изменение любого из них начинает новый префикс кэша. Переключение между `adaptive`, `enabled` и `disabled`, изменение `budget_tokens` и изменение значения усилий сбрасывают точки кэширования: точки на уровне сообщений всегда промахиваются, а точки на инструментах и системном промпте могут промахиваться тоже, в зависимости от того, куда модель вкладывает конфигурацию. Считайте любое изменение мышления или усилий началом кэша заново. Идущие подряд запросы с одинаковой конфигурацией кэш сохраняют, а явная установка параметра в значение по умолчанию эквивалентна его отсутствию.

Конфигурация вкладывается в промпт, поэтому её смена рвёт префикс

Блоки мышления кэшируются вместе с результатами инструментов. Внутри цикла работы с инструментами кэширование происходит, когда вы делаете следующий запрос с результатами. В этот момент предыдущая история диалога, включая блоки мышления, может быть закэширована, и эти закэшированные блоки считаются входными токенами в вашей статистике при чтении из кэша. Это происходит автоматически, даже без явных маркеров `cache_control`, и одинаково работает для обычного и чередующегося мышления. Плата за это: блоки мышления, которых вы больше никогда не увидите в ответах, всё равно вносят вклад в расход входных токенов при чтении из кэша.

Есть ли прошлые блоки в контексте вообще, зависит от модели. На моделях, которые хранят всё, блоки мышления прошлых ходов остаются в кэше и в контексте. На моделях, которые хранят только последний ход, как только вы отправляете пользовательское сообщение, отличное от результата инструмента, все предыдущие блоки мышления вырезаются из контекста. На таких моделях вот такой диалог:

ИЗ ДОКУМЕНТАЦИИ

```
User: ["What's the weather in Paris?"],
Assistant: [thinking_block_1] + [tool_use block 1],
User: [tool_result_1, cache=True],
Assistant: [thinking_block_2] + [text block 2],
User: [Text response, cache=True]
```

обрабатывается так, как будто блоков мышления там не было вовсе:

ИЗ ДОКУМЕНТАЦИИ

```
User: ["What's the weather in Paris?"],
Assistant: [tool_use block 1],
User: [tool_result_1, cache=True],
Assistant: [text block 2],
User: [Text response, cache=True]
```

На моделях, которые хранят всё, тот же запрос удерживает `thinking_block_1` и `thinking_block_2` и в контексте, и в кэше.

Деградация вырезает мышление из кэшируемой истории. Если мышление выключается посреди хода и вы передаёте контент мышления в текущем ходе с инструментами, этот контент вырезается, а мышление остаётся выключенным для этого запроса. Чередующееся мышление усиливает эффект сброса кэша, поскольку блоки мышления могут возникать между несколькими вызовами инструментов.

Задачи с большим объёмом мышления часто идут дольше, чем стандартное время жизни кэша в 5 минут. Имеет смысл рассмотреть часовое время жизни кэша, чтобы удерживать попадания на длинных сессиях мышления и многошаговых сценариях.

Мышление и контекстное окно

`max_tokens`, который включает всё мышление, сгенерированное в текущем ходе, применяется как строгий лимит. На моделях Claude 4.5 и новее, если входные токены плюс `max_tokens` превышают размер контекстного окна, API принимает запрос. Если генерация затем упирается в лимит окна, она останавливается со `stop_reason: "model_context_window_exceeded"`, а не возвращает ошибку. На более ранних моделях API вместо этого вернёт ошибку валидации.

Как мышление считается против окна, решает момент его появления.

Мышление текущего хода всегда считается в `max_tokens`, оплачивается как выходные токены и занимает место в контекстном окне того хода, который его породил.

Мышление прошлых ходов зависит от режима сохранения. На моделях, которые хранят все прошлые ходы, предыдущие блоки остаются в контексте, считаются в окно и оплачиваются как входные токены, как и вся остальная история диалога. На моделях, которые хранят только последний ход, API вырезает старые блоки автоматически, и они не занимают ни места в окне, ни входных токенов.

На практике: на моделях, которые хранят всё, планируйте контекстное окно так, как будто мышление это обычная история диалога, потому что это она и есть. Длинные агентные сессии накапливают мышление в контексте, и если нужно вернуть себе место, применяйте очистку блоков мышления через редактирование контекста. На моделях, которые хранят только последний ход, мышление стоит ровно один ход: мышление каждого хода считается против `max_tokens` этого хода и затем выпадает из окна.

Чтобы получить точные числа под свой сценарий, особенно для многоходовых диалогов с мышлением, используйте API подсчёта токенов.

Шифрование мышления

Полный контент мышления шифруется и возвращается в поле `signature` каждого блока. API использует подпись, чтобы проверить, что блоки мышления были сгенерированы Claude, когда вы передаёте их обратно.

Что важно помнить про подписи:

- Строго обязательно возвращать блоки мышления только при использовании инструментов вместе с мышлением. В остальных случаях блоки прошлых ходов можно опускать.
- Возвращая блоки, передавайте всё ровно в том виде, в каком получили.
- При стриминге подпись приходит как `signature_delta` внутри события `content_block_delta` прямо перед событием `content_block_stop`.
- Значения `signature` заметно длиннее на моделях Claude 4 и новее, чем на предыдущих.
- Поле `signature` непрозрачно: не пытайтесь его интерпретировать или разбирать.
- Значения `signature` совместимы между платформами. Значения, сгенерированные на одной платформе, работают на другой.

Блоки, скрытые по безопасности

Помимо обычных блоков `thinking`, API может возвращать блоки `redacted_thinking`, когда часть рассуждений Claude скрыта фильтром безопасности. В переводе они называются скрытыми по безопасности: содержимое зашифровано и вам не показано, но сами блоки никто не правил. Это не то же самое, что пустое поле при `display: "omitted"`. Такой блок содержит зашифрованный контент в поле `data` и не имеет читаемого текста:

ИЗ ДОКУМЕНТАЦИИ

```
{
  "type": "redacted_thinking",
  "data": "..."
}
```

Поле `data` непрозрачно и зашифровано. Как и поле `signature` обычных блоков, блоки `redacted_thinking` нужно передавать обратно в API без изменений при продолжении многоходового диалога с инструментами.

Если ваш код фильтрует блоки контента по типу, например через `block.type == "thinking"`, при возврате ответов с инструментами включайте туда и `redacted_thinking`. Фильтрация только по `block.type == "thinking"` молча теряет такие блоки и ломает многоходовой протокол.

Блоки `redacted_thinking` это отдельный тип блока, возвращаемый, когда мышление скрыто фильтром безопасности. Это не то же самое, что опция `display: "omitted"`, которая возвращает обычные блоки `thinking` с пустым полем.

ЧАСТЬ · V

Границы и разбор

Особенности Fable и Mythos, матрица ограничений, разбор от канала и первоисточники.

Вывод мышления на Fable 5 и Mythos 5

На Claude Fable 5 и Claude Mythos 5 полные рассуждения не возвращаются никогда. Блоки, которые вы получаете, это обычные блоки `thinking`, а не `redacted_thinking`, и настройка `display` работает так же, как на других моделях: текст сводки либо пустое поле `thinking` при значении `omitted`, которое на этих моделях стоит по умолчанию.

Продолжая диалог на той же модели, передавайте каждый блок мышления обратно ровно в том виде, в каком получили, включая блоки с пустым полем `thinking`. Не редактируйте и не реконструируйте их. Читать текст сводки для отображения нормально: API отклоняет блоки, содержимое которых было изменено, а не блоки, которые вы прочитали.

Два исключения описаны в разделе про fallback-кредит: повторные запросы по fallback-кредиту обязаны дословно повторять тело отклонённого запроса, а блоки `fallback` от срабатывания посреди вывода остаются там, где появились.

Чтобы получить видимость рассуждений модели, читайте блоки `thinking`, а не просите модель изложить рассуждения в тексте ответа. На Claude Fable 5 запрос, который пытается вытянуть внутренние рассуждения модели в составе текста ответа, может быть отклонён с `stop_details.category: "reasoning_extraction"`.

Ограничения и совместимость

1M	128k	64k	300k	21 333
контекстное окно	выход, 5-е поколение	выход, Haiku 4.5 и Opus 4.5	batch с бета-заголовком	порог, выше которого SDK требует стриминг

Лимиты, которые упираются в мышление

	5-е поколение	более старые
temperature, top_p, top_k	400 всегда	400 только при мышлении
Предзаполнение ответа	нельзя при мышлении	нельзя при мышлении
Принудительный вызов инструмента	работает в adaptive	не работает в ручном режиме

Что перестаёт работать при включённом мышлении

Параметры сэмплирования. На Claude Fable 5, Claude Mythos 5, Claude Mythos Preview, Claude Opus 5, Claude Opus 4.8, Claude Opus 4.7 и Claude Sonnet 5 значения `temperature`, `top_p` или `top_k`, отличные от стандартных, возвращают ошибку 400 на каждом запросе, независимо от того, используется ли мышление. На более старых моделях ограничение действует только пока мышление включено: `temperature` и `top_k` с мышлением несовместимы, а `top_p` допустим в значениях между 0.95 и 1.

Предзаполнение ответа и принудительный вызов инструментов. Предзаполнить ответ ассистента при включённом мышлении нельзя. Принудительный вызов инструментов несовместим с ручным extended thinking, но работает с адаптивным мышлением.

Лимиты вывода. Claude Fable 5, Claude Mythos 5, Claude Mythos Preview, Claude Opus 5, Claude Opus 4.8, Claude Opus 4.7, Claude Sonnet 5, Claude Opus 4.6 и Claude Sonnet 4.6 поддерживают до 128k выходных токенов на запрос. Claude Haiku 4.5, Claude Sonnet 4.5 и Claude Opus 4.5 поддерживают до 64k. На Message Batches API бета-заголовок `output-300k-2026-03-24` поднимает лимит до 300k для Claude Opus 5, Claude Opus 4.8, Claude Opus 4.7, Claude Sonnet 5, Claude Opus 4.6 и Claude Sonnet 4.6.

Длинные запросы. SDK требуют стриминга, когда `max_tokens` больше 21 333, чтобы избежать таймаутов HTTP на долгих запросах. Это клиентская валидация, а не ограничение API. Если обрабатывать события по одному не нужно, используйте `.stream()` с `.get_final_message()` в Python или `.finalMessage()` в TypeScript. Ждите более долгих ответов при активном мышлении. Для нагрузок, где мышление превышает примерно 32k токенов на запрос, используйте пакетную обработку: такие запросы могут идти достаточно долго, чтобы упереться в системные таймауты и лимиты открытых соединений.

Разбор от канала

Перевод закончен. Дальше моё.

Пять выводов

1. **Extended thinking стал legacy, и это главная новость страницы.** Параметр `budget_tokens`, вокруг которого построены все прошлогодние гайды и половина обвязок, теперь обслуживает только модели, которые не умеют иначе. Актуальная конфигурация это `type: "adaptive"` плюс `effort`.

2. **Вы платите за мышление, которого не видите.** На всех моделях пятого поколения `display` по умолчанию стоит в `omitted`. Пустое поле `thinking` в ответе означает не «модель не думала», а «модель думала, вы заплатили, текст вам не отдали». Отдельно сказано прямым текстом: так экономится время, но не деньги.

3. **Полных рассуждений не отдаёт ни одна настройка.** Даже `summarized` возвращает пересказ, которую делает другая модель, и модель, которая думала, эту сводку не видит. Число оплаченных выходных токенов не совпадает с тем, что вы видите в ответе.

4. **Любое изменение мышления или усилий обнуляет кэш промпта.** Конфигурация встраивается в сам промпт. Для агента, который динамически крутит `effort` между шагами, это означает промах кэша на каждом переключении. Дешёвый на вид рычаг экономии оказывается дорогим.

5. **На новых моделях мышление копится в контексте.** Opus начиная с 4.5, Sonnet начиная с 4.6, а также Fable 5, Mythos 5 и Mythos Preview хранят блоки всех прошлых ходов, и они оплачиваются как входные токены наравне с историей. Длинная агентная сессия платит за своё мышление дважды: один раз как за выход, потом многократно как за вход.

Как это выглядит на фоне остальных

Самое интересное в этой странице видно только в сравнении. Я сходил в документацию OpenAI и Google и сверил пять пунктов.

Сырые рассуждения не отдаёт никто. У OpenAI reasoning-токены «не видны через API», доступна только сводка. У Google возвращаются thought summaries, а не сами мысли. У Anthropic ни одна настройка `display` не отдаёт полную цепочку. Три независимые лаборатории закрыли одно и то же в одно и то же время.

	Anthropic	OpenAI	Google
Сырые рассуждения	только сводка	только сводка	только сводка
Оплата токенов	как выходные	как выходные	как выходные
Рычаг управления	effort	reasoning.effort	thinking_level
Выключить совсем	не везде можно	есть none	нельзя
Подпись блока	signature	encrypted_content	thought signature

Три лаборатории независимо пришли к одной конструкции

Платят за них тоже везде одинаково: как за выходные токены. Формулировка Google дословно: цена ответа это сумма выходных токенов и токенов мышления.

Рычаг управления везде превратился из числа в шкалу. У OpenAI `reasoning.effort` со значениями от `none` до `max`, у Google `thinking_level` от `minimal` до `high`, у Anthropic `effort` от `low` до `xhigh`. Бюджет в токенах, вокруг которого строились прошлогодние гайды, вымылся у всех троих.

Выключить мышление совсем становится нельзя. У Google на текущих моделях выключить нельзя вообще, у Anthropic нельзя на Fable, Mythos и на верхних уровнях усилий Opus 5. Из троих полный выключатель остался только у OpenAI.

И подпись везде обязательно передавать обратно: `signature` у Anthropic, `encrypted_content` у OpenAI в stateless-режиме, thought signature у Google с капслоком «MUST» в документации.

Арифметика длинной сессии

Из правил документа следует вещь, которую он сам не проговаривает.

На моделях, которые хранят все прошлые ходы, мышление каждого хода остаётся в контексте и на каждом следующем ходу оплачивается заново, уже как входные токены. Значит расход на мышление растёт не линейно, а квадратично от числа ходов.

Считаем. Сессия на 20 ходов, в каждом ходу примерно 2000 токенов мышления. Выходных токенов мышления за сессию: 40 000. А входных, за счёт того, что каждый прошлый блок перечитывается на каждом следующем ходу: около 380 000. В девять с половиной раз больше.

Вход дешевле выхода, поэтому в деньгах разрыв не девятикратный. Но порядок расхода задаёт именно эта сумма, а не то, что вы видите в ответах. И растёт она как квадрат длины сессии.

Инструмент против этого в документе назван: очистка блоков мышления через редактирование контекста. Упомянут вскользь, тремя ссылками, без единого примера.

Три ставки

Это уже мои прогнозы, не документ. Ставлю на них с разной уверенностью.

Сырые рассуждения закрыты навсегда, уверенность высокая. Три лаборатории закрылись синхронно, и причина у всех одна: по цепочке рассуждений конкурента можно дистиллировать модель. Это не продуктовое решение, которое отыграют назад по просьбам разработчиков.

Ручное управление бюджетом умрёт полностью, уверенность средняя. Сначала `budget_tokens` уступил шкале усилий. Логичный следующий шаг это исчезновение и шкалы: модель сама решает, сколько думать, а разработчик задаёт только потолок расходов через `max_tokens`. Adaptive это уже наполовину так.

Мышление станет неотключаемой частью инференса, уверенность средняя. Два вендора из трёх уже отобрали выключатель на части моделей. Если тренд продолжится, вопрос «думать или нет» исчезнет из API, останется только «насколько глубоко».

Что из этого следует практически: не стройте продукт на предположении, что вы увидите рассуждения модели, и не стройте бюджет на том, что видно в ответах. Смотреть надо на счётчики использования, а не на длину текста.

Что проверить в своём коде сегодня

- Фильтруете ли вы блоки по `block.type == "thinking"` при возврате результатов инструментов. Если да, вы молча теряете `redacted_thinking` и ломаете протокол.
- Переключаете ли вы модель посреди диалога. Если да, блоки мышления прошлых ходов надо вырезать: чужие модели их не отклонят, но входные токены за них спишут.
- Стоит ли у вас `temperature` не по умолчанию. На моделях пятого поколения это ошибка 400 на каждом запросе, независимо от мышления.
- Превышает ли `max_tokens` значение 21 333. Тогда SDK потребует стриминг.
- Меняете ли вы `effort` между запросами внутри одной сессии. Каждое изменение начинает кэш заново.

Чего в документе нет

Страница не перечисляет уровни усилий и не называет уровень по умолчанию, а отправляет за этим на две другие страницы. Нет ни одной цифры про то, насколько мышление улучшает результат, только слово «улучшает». Нет цен: сколько стоят токены мышления относительно обычных выходных, надо смотреть отдельно.

И нет ответа на главный вопрос эксплуатации: как понять, что модель думала слишком много для этой задачи. Единственный доступный сигнал это счёт.

Источники

Перевод сделан с англоязычной страницы Thinking в документации Anthropic. Перевод неофициальный: при любом расхождении верен оригинал. Ссылки кликабельны.

ПЕРВОИСТОЧНИКИ

1. **Anthropic · Thinking (adaptive thinking)** · platform.claude.com
2. **Anthropic · Extended thinking (legacy)** · platform.claude.com
3. **Anthropic · Steering thinking and cost** · platform.claude.com
4. **Anthropic · Thinking in tool and multi-turn workflows** · platform.claude.com
5. **Anthropic · Troubleshooting thinking** · platform.claude.com
6. **Anthropic · Effort** · platform.claude.com
7. **Anthropic · Prompt caching** · platform.claude.com
8. **Anthropic · Context editing (thinking block clearing)** · platform.claude.com
9. **Anthropic · Streaming Messages** · platform.claude.com
10. **Anthropic · Messages API reference** · platform.claude.com

ПОД КАПОТОМ · 2026

Спасибо, что дочитал.

Перевод страницы Thinking для канала «Under the Hood»: разборы ИИ-индустрии без хайпа, с источниками. Если пригодилось, забирай продолжение.

Under the Hood

@gorilla_under_hood

Перевод	Илья Утов
Telegram	**@wh_zt**
LinkedIn	**linkedin.com/in/ilyautov**