ローカルLLMで議事録は作れるか 2026年7月版

64GB Mac mini・27Bモデルの実用性評価：Thinking税・コンテキスト上限・フロンティアモデル比較

Can Local LLMs Take the Minutes? [July 2026 Edition] — A Practicality Evaluation of 27B-Class Models on a 64GB Mac mini: The Thinking Tax, Context Ceilings, and a Frontier-Model Reference

著者	大橋 功	所属	株式会社喋ラボ	公開日	2026年7月
測定日	2026年7月18〜19日	シリーズ	喋ラボ テクニカルペーパー（第2弾）（前作: LLM量子化ベンチマーク）		

ABSTRACT

64GB Mac mini（Apple M4 Pro）上で、Thinkingモード（答える前に思考を挟むモード）を測定した。 対象は 4B / 9B / 27B の3サイズ、4モデル（qwen3.5:4b / 9b / 27b、qwen3.6:27b）。 総時間・生成トークン・出力品質への影響を定量化している。

品質の比較対象として、フロンティアモデル7種も同じデータセットで測定した （Claude Fable 5 / Opus 4.8 / Sonnet 5、GPT-5.5 / 5.4 / 5.4-mini / 5.1）。

主要な結果は次のとおり。

1. **Thinkingは時間を4.7〜8.7倍に膨張させる。** 税率はモデルが小さいほど高い。
2. **より重要なのは、その代償に対して成果物がゼロになることである。** num_ctx 32768 で23,403トークンの会議録を処理すると、qwen3.5系の3モデルはすべて コンテキスト上限まで思考し続けて打ち切られた。**JSON有効率0〜20%、F1中央値0.000**。 時間を6〜9倍払った上で、抽出結果が一つも得られない。
3. **例外は qwen3.6:27b のみ。** 同じ条件で思考を7,287〜7,975トークンで自然に切り上げ、 5試行中4試行が成立した（F1中央値0.781）。「思考を切り上げる能力」に世代間の差がある。
4. **ローカル最良は qwen3.6:27b**（Thinkingオフ、F1 0.788）。 これはフロンティア最上位（Claude Fable 5、F1 0.932）の約86%にあたり、 **GPT-5.4-mini（0.677）を上回る**。
5. **Thinkingの恩恵はモデルが弱いほど大きい。** F1の変化は GPT-5.4-mini が +0.20、GPT-5.1 が +0.06、Claude Sonnet 5 が +0.01、 Claude Opus 4.8 は **−0.02**。例外のない順序になっている。 強いモデルを使えるならThinkingは要らず、弱いモデルほど補正が効く。

あわせて、機密ゼロで公開可能な**正解ラベル付き日本語会議データセット**（37項目、CC BY 4.0）を構築・公開する。

1. 背景と問い

前作では、64GB Mac mini でローカルLLMを実運用する際の落とし穴を2つ報告した。 巨大なデフォルトコンテキストによるメモリ膨張と、Thinkingモードの暴走である（後者は300字要約に最大12分20秒、8,679トークンを要した）。

本稿はこの2つ目を測り直す。**一度きりの逸話ではなく、モデルサイズを横断する性質として 定量化する**ことが目的である。

問いは3つ。

1. **Thinking税はどれだけか。** 総時間と生成トークンがどれだけ膨張し、それはサイズによってどう変わるか。
2. **その税に見合う品質向上はあるか。** 特に「推論を要する項目」で正答率が上がるのか。
3. **ローカルモデルは実務で使えるか。** 議事録生成・構造化データ抽出という具体的用途で、 どのサイズなら何ができるのか。

3つ目に答えるため、フロンティアモデル（Claude / GPT）を**品質の参照軸**として追加した。 これは単なる比較のためではない。ローカルモデルだけを測っていると、 低いスコアが「モデルの限界」なのか「問題が曖昧で誰にも解けない」のかを切り分けられない。 フロンティアモデルは**正解ラベルの妥当性検証**として機能する（§7.3で実際に機能した）。

---

2. 実験環境

項目	値
ハードウェア	Mac mini (Mac16,11) / Apple M4 Pro / 12コア
ユニファイドメモリ	64 GB
OS	macOS 26.5.1 (25F80)
推論基盤	Ollama 0.30.10（HTTP API `/api/chat`）
コンテキスト長	`num_ctx: 32768` 固定（追加測定で65536）
測定モデル	qwen3.5:4b / qwen3.5:9b / qwen3.5:27b / qwen3.6:27b
試行数	各条件5試行、代表値は中央値

`qwen3.5:27b` は GGUF / Q4_K_M / 27.8Bパラメータ / 17GB。

すべての測定は Ollama の HTTP API 経由で行った。自動化と再現性のため、対話CLIは使っていない。

Thinkingの制御はリクエストの `think: true/false` で行う。 このパラメータが実際に効くことは、事前に小さいモデルで確認した。 `think: false` ではレスポンスに `thinking` フィールドが現れず、 `think: true` では思考過程が生成される。

2.1 用語

本稿で使う評価用語を先に定義する。機械学習の評価に馴染みのない読者向け。

用語	意味
gold（ゴールド）	**正解データ。** "gold standard"（基準となる正解）の略。本稿では設計した37項目の正解を指し、`dataset/gold.json` に格納している。モデルの出力をこれと突き合わせて採点する
recall（再現率）	正解37項目のうち、モデルがいくつ取れたか。**取りこぼしの少なさ**
precision（適合率）	モデルが出力した項目のうち、いくつが正しかったか。**余計なものを出さない度合い**
F1	recall と precision の調和平均。両方をまとめた指標で、0〜1。1に近いほど良い。片方だけ高くても F1 は上がらない
捏造（英語では hallucination）	正解にも許容リストにもない項目をモデルが出力すること。会議で誰も言っていないことを議事録に書く行為にあたる
許容リスト	正解ではないが、抽出されても妥当な記述。加点も減点もしない。これを設けないと、正しい抽出をしたモデルを不当に罰することになる（§3.4を参照）
num_ctx	コンテキスト長。モデルが一度に扱えるトークン数の上限。**入力と出力の合計**に効く
トークン	モデルが文章を扱う単位。日本語では概ね 1文字 ≒ 0.7トークン（本データセットでの実測値）
tok/s	1秒あたりの生成トークン数。生成の速さ
中央値	5試行を小さい順に並べた真ん中の値。極端な外れ値に引きずられないため、本稿の代表値として用いる

3. データセット

3.1 設計方針

実在企業の会議データは公開できない。だからこそ**機密ゼロで公開可能な正解ラベル付きデータ**を　構築する必要がある。方式は「先に正解を設計し、それが埋まるようにトランスクリプトを逆算して書く」。

項目	値
本文	33,401文字 / 668行 / **23,403トークン**
会議	架空のミドリ精機株式会社「プロジェクト・アオゾラ」2026年5月13日(水) 09:30〜15:50
話者	6名（全て架空）
正解項目	**37**（decisions 12 / action_items 12 / key_numbers 13）
ライセンス	CC BY 4.0

3.2 難易度タグ

各項目に、何が難しさの源かを示すタグを付けた。

タグ	数	定義	分かれる想定
`easy`	14	明示的な発言から直接取れる	全モデルが取れて当然の下限
`medium`	7	少し文脈が要るが、近くを読めば分かる	—
`hard_size`	5	文書の離れた箇所を参照する、後から訂正された内容を追う	モデルサイズ／実際に扱える文脈の長さ
`hard_thinking`	7	複数ステップの計算、条件分岐、消去法、相対日付の変換	Thinkingの有無
`hard_implicit`	4	決定を表す言葉が出てこない、誰が担当か分かりにくい	どちらにも依存しにくい

`easy` は測定が健全かどうかを確かめるための下限である。**フロンティアモデルが `easy` を落とすなら、 それはモデルではなく問題を疑うべき**という切り分けに使う。

`hard_implicit` は当初存在しなかった。独立検証で次の指摘を受け、新設して分離した。 「`hard_size` に分類した4項目は、実際には離れた箇所を見比べる必要がない。 難しさの源は、決定を表す言葉が出てこないことにある」

3.3 仕掛けの例

- **複数ステップの計算**: 「今期は前回の1.2倍」（13:00台）＋「前回のシステム予算の枠は500万」（09:39） → 正解600万円。**600という数字は本文に一度も現れない**
- **離れた箇所への参照**: 09:36に「応答時間1.2秒以内」→ 14:00台に「さっき言ってた目標値」とだけ照応
- **訂正・上書き**: 09:35「9月末リリース」→ 15:00台「十月十五日に倒しましょう」。最終状態が正解
- **暗黙の合意**: 「じゃあ、それで。」だけで決定が成立。「決定します」「承認します」といった言葉が一切出てこない
- **ノイズ**: 昼休憩区間に無関係な数値（定食屋850円 / 気温28度 / 電車の遅延12分 / 子供5歳）を配置。 これらを重要数値として拾った場合は明確な捏造としてカウントする

3.4 独立検証

執筆に関与していない検証者に3回の検証を依頼し、**うち2回で NO-GO 判定**が出た。 発見された欠陥はいずれも設計者自身では気づけないものだった。

#	欠陥	対応
1	難易度分類そのものが誤り（`hard_size` の半分が別種の難しさ）	`hard_implicit` を新設
2	計算を要する1問で答えが1つに定まらなかった（2つの「前回」が同じものを指す保証がなかった）	本文修正
3	「担当者の取り違え」を狙った仕掛けが機能せず（議長の復唱で3重に確定していた）	復唱を削除
4	goldが網羅的でなく、正しい抽出を誤答扱いする状態	許容リスト53件を導入
5	不当な引っかけの混入（正解と同軸の紛らわしい数値）	削除
6	`decided_by` の採点基準が不統一	複数正解を許容

教訓: 正解ラベル付きデータセットを自作する場合、設計者自身では欠陥を見つけられない。

4. 採点方法

4.1 基本

- **JSON有効率**: コードフェンス除去等の前処理を一律適用した上でパース可能か
- **項目別F1**: precision の分母から「許容項目」を除外する。本文中には gold の正解ではないが 抽出されても妥当な記述が存在し、これを分母に含めると**正しい抽出をしたモデルを不当に罰する**
- **捏造率**: goldにも許容リストにもマッチしない出力。ノイズ区間の数値を拾えば明確な捏造
- **難易度タグ別正答率**: 考察の核

正誤は識別テキストのマッチだけでは決めず、`decided_by` / `assignee` / `due` / `value` の フィールド照合まで要求する。仕掛けはまさにこれらのフィールドで正誤が分かれるよう設計されている。

4.2 実用グレード

F1だけでは「業務投入の可否」に答えられない。同じF1でも失敗の種類で深刻さが違うためである。

失敗の種類	議事録として	外部システム連携として
JSONが壊れる	手直しすれば使える	連携が止まる。致命的
決定事項の取りこぼし	議事録として失格	影響は限定的
存在しない項目の捏造	信用を失う	ゴミデータが流入。最悪
担当者・期限の誤り	参加者が気づく	誤ったタスクが割り当たる。致命的

グレード	条件
A（自動連携可）	JSON有効 / easy・medium全問正解 / 捏造ゼロ / 担当者・期限の誤りゼロ
B（人手レビュー前提）	JSON有効 / easy全問正解 / 捏造ゼロ。取りこぼしは許容
C（実務投入不可）	JSONが壊れる、easyを落とす、または捏造がある

4.3 【重要】採点系そのものの検証

フロンティアモデルで疎通確認した際に異常に低いスコアが出て、**採点ロジックに4つの欠陥**が あることが判明した。最も深刻だったのは、`difflib.SequenceMatcher` の文字列類似度が **日本語で機能していなかった**ことである。

```
gold「移行方式はB案（段階移行）を採用する」
出力「移行方式は拠点を分けて順次切り替える段階移行（B案）で確定する」→ 類似度0.49で未検出扱い
gold「目標応答時間」 vs 出力「主要画面の応答時間目標」                  → 類似度0.47で未検出扱い
```

日本語は単語境界が無いため、内容が正しくても語順が違う・修飾が付くだけで類似度が急落する。 結果として recall と precision の両方が過小評価され、**丁寧に書くモデルほど不当に罰せられていた**。

対策として、文字bigramのDice係数と一方が他方を含む度合いを併用し、カテゴリを跨ぐ対応付けと 値ベースの許容判定を追加した。修正の効果（Claude Sonnet 5 / 1試行）：

	修正前	修正後
F1 (off/on)	0.692 / 0.800	**0.892 / 0.904**
捏造	10 / 5	**0 / 0**
easy	0.93	**1.00**

F1が0.2動いた。 この修正前に本測定を採点していれば、「ローカルもフロンティアも大したことがない」という誤った結論を出していた。

教訓: 自動採点系そのものを検証しないと、測っているのはモデルの性能ではなく採点系のバグである。 フロンティアモデルは品質の参照軸としてだけでなく、**採点方法が妥当かを確かめる手段としても機能した**。

---

5. Thinking税：本稿の主結果

5.1 時間とトークン

各条件5試行、代表値は中央値。プロンプトは全モデル同一（23,403トークンの会議録＋抽出指示）。

モデル	時間 off	時間 on		トークン off	トークン on	倍率
qwen3.5:4b	24.7s	215.9s	**8.73倍**	1,005	9,069	9.02倍
qwen3.5:9b	47.1s	297.9s	**6.33倍**	1,469	9,069	6.17倍
qwen3.5:27b	151.7s	865.3s	**5.70倍**	1,604	9,069	5.65倍
qwen3.6:27b	156.9s	733.6s	**4.68倍**	1,673	7,725	4.62倍

税率はモデルが大きいほど小さく出ているが、**これは「大きいモデルほどThinkingが軽い」という意味ではない**。 生成トークンの上限が全モデル共通のため、素の生成が遅い27Bほど倍率が圧縮されて見えるだけである。 **絶対時間で見れば27Bが最悪の14分25秒**であり、この解釈には注意を要する。

5.2 生成速度は長文脈で落ちる

モデル	前作（短いプロンプト）	本測定（23.4kトークン）
4B	54 tok/s	**44.1 tok/s**
9B	36 tok/s	**31.4 tok/s**
27B	12 tok/s	**10.8 tok/s**

いずれも1〜2割低下している。**長文脈では生成速度そのものが落ちる**。 カタログ値や短文での実測をそのまま長文脈の見積もりに使うと、所要時間を過小評価する。

5.3 【核心】コンテキスト上限への張り付き

`eval_count` を見ると、**qwen3.5系の3モデルは毎回きっかり9,069トークン**を生成している。 これは偶然ではない。

```
プロンプト 23,699 + 生成 9,069 = 32,768 = num_ctx ちょうど
```

つまり**ばらつきではなく、コンテキスト上限まで思考し続けて強制的に打ち切られている。**

モデル	上限到達	生成トークンの範囲
qwen3.5:4b	**5/5**	9,069（全試行同一）
qwen3.5:9b	**4/5**	9,069 ×4／4,496 ×1
qwen3.5:27b	**5/5**	9,069（全試行同一）
qwen3.6:27b	**1/5**	**7,287 / 7,615 / 7,725 / 7,975** / 9,069

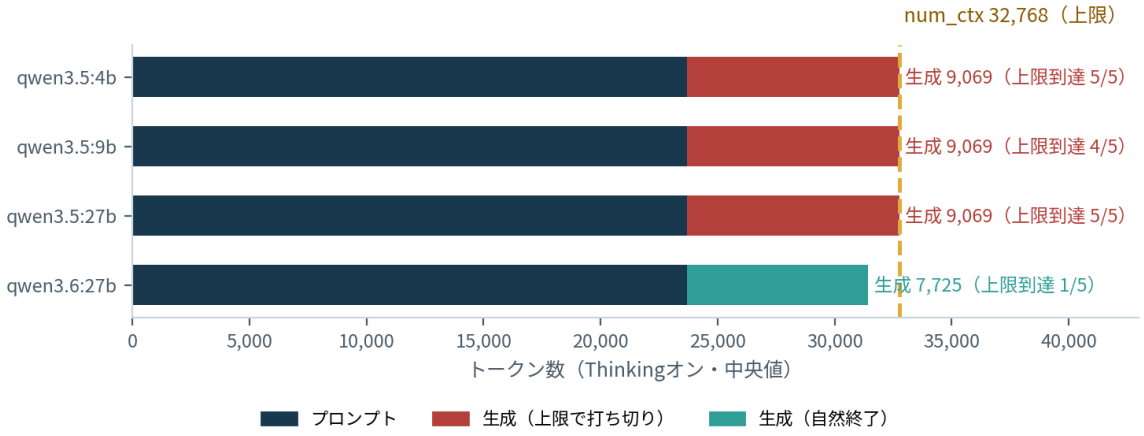

図1　Thinkingオン時のコンテキスト消費 — qwen3.5系は上限32,768に張り付き、qwen3.6のみ自然終了

qwen3.6:27b だけが思考を自然に切り上げている。 同じ27Bでも世代が違えば挙動が変わる。「残りコンテキストを意識して思考を打ち切る能力」が qwen3.6 で獲得されたと解釈できる。

ただし qwen3.6 の合計も 30,986〜31,674 で、上限まで残り1,000〜1,800トークンしかない。 **かろうじて収まっているだけで、会議録がもう少し長ければ破綻する。** 実務上は危険域である。

5.4 num_ctx 65536：Thinkingは「壊れて」いない、枠が足りなかっただけ

§5.3の観察からは2つの解釈があり得た。

- **仮説X**: 単に枠が足りない。もっと与えれば思考は完走する
- **仮説Y**: 止まり方を知らない。与えた分だけ消費し、枠を増やしても壁が移動するだけ

仮説Yを疑う根拠はあった。qwen3.5:4b と 27b は15試行中14回で `eval_count` がきっかり9,069であり、 自然な終点があるならばらつくはずだからである。 これを判定するため、最も安く決着がつく4Bで num_ctx を倍にして測定した。

結果は仮説Xの完全な支持である。

num_ctx	生成トークン中央値	生成トークンの範囲	時間中央値	上限到達	自然終了
32,768	9,069	9,069（全試行同一）	215.9s	**5/5**	**0/5**
65,536	**12,315**	**10,904〜15,399**	305.7s	**0/5**	**5/5**

qwen3.5:4b が必要としていたのは 10,904〜15,399トークンであり、9,069では足りなかった。 枠を与えれば**全試行が `done_reason=stop` で自然終了**する。 生成量がばらついていること自体が、上限張り付きではなく自然な終了であることの証拠である。

したがって「ローカルモデルのThinkingは壊れている」という理解は誤りである。 正しくは「**num_ctx 32768 という設定が、23.4kトークンの文書に対して不足していた**」。

ただし代償がある。時間は 215.9s → 305.7s（**1.42倍**）に増える。 Thinkingオフ（24.7s）と比較すれば **12.4倍**であり、税そのものは消えない。

実務上の含意: 長い文書に対してThinkingを使うなら、 **num_ctx はプロンプト長の2.5〜3倍を確保すべきである。** 本測定では 23.4k のプロンプトに対し、32.8k（1.4倍）では足りず、65.5k（2.8倍）で成立した。 64GB機ならメモリに余裕があるため、この設定は現実的である。

> **注記**: 本測定は4Bのみである。27Bで同様に完走するか、 またその際のメモリ増がどれだけかは未測定であり、次回の課題とする。 4Bが必要とした12,315トークンに対し、27Bがどれだけ必要とするかは自明でない。

---

6. 品質評価：ローカルモデル

num_ctx 32768、各条件5試行、中央値。

モデル	Thinking	JSON有効	出力切れ	F1中央	easy	medium	hard_size	hard_thinking	hard_implicit
4B	off	5/5	0	0.431	0.57	0.00	0.60	0.00	0.00
4B	**on**	**0/5**	**5**	**0.000**	—	—	—	—	—
9B	off	5/5	0	0.561	0.71	0.14	0.60	0.14	0.50
9B	**on**	**1/5**	**4**	**0.000**	—	—	—	—	—
27B(3.5)	off	4/5	0	0.635	0.86	0.29	0.60	0.29	0.50
27B(3.5)	**on**	**0/5**	**5**	**0.000**	—	—	—	—	—
27B(3.6)	off	5/5	0	**0.788**	0.93	0.57	0.80	0.29	0.75
27B(3.6)	**on**	4/5	1	**0.781**	0.93	0.57	0.60	**0.43**	0.75

Thinkingは税ではなく没収である。 qwen3.5系は時間を6〜9倍払った上でF1が0.000になる。 唯一 qwen3.6:27b が成立し、 `hard_thinking` が 0.29→0.43 と改善して**全40試行で唯一のグレードB**を1回記録した。

注目すべきは **qwen3.5:27b がThinkingオフでも5回に1回JSONを壊している**ことである。 出力形式の安定性という観点では、qwen3.6:27b が明確に優れている。

グレードAは全40試行でゼロ。 現時点で「議事録を自動で外部システムへ流す」用途に耐える ローカルモデルは、この構成では存在しない。

---

7. フロンティアモデルとの比較

7.1 位置づけ

フロンティアモデルは**品質軸のみ**に参加させ、時間軸には載せない。　　ネットワーク遅延・共有インフラ・非公開のハードウェア構成が混入し、 64GB Mac miniでの実測と同じ土俵にならないためである。

比較可能性のため、**structured outputs（JSON Schema強制）は使っていない。** 使えばJSONが必ず妥当になり、JSON有効率をローカルモデルと同じ土俵で測れなくなる。 また既定値がモデルごとに異なるため、Thinkingのon/offは常に明示指定した。

なお **Claude Fable 5 は思考を無効化できない。** `thinking: {"type":"disabled"}` を送るとエラーになるため、Thinkingオフは原理的に測れない。 推定値で埋めず、「測定不可」として記録している。

7.2 結果

モデル	Thinking	F1中央	easy	medium	hard_size	hard_thinking	グレード
Claude Fable 5	on（固定）	**0.932**	1.00	0.86	1.00	**0.71**	B×5
GPT-5.5	off	**0.904**	1.00	**1.00**	1.00	**0.71**	C×5
Claude Sonnet 5	on	0.904	1.00	0.71	1.00	0.71	B×4, C×1
Claude Opus 4.8	off	0.904	1.00	**1.00**	1.00	0.57	**A×3**, C×2
GPT-5.5	on	0.892	1.00	**1.00**	1.00	0.57	**A×1**, C×4
Claude Sonnet 5	off	0.892	1.00	0.86	1.00	0.57	B×3, C×2
Claude Opus 4.8	on	0.889	1.00	**1.00**	0.80	0.43	**A×3**, C×2
GPT-5.4-mini	on	0.880	1.00	0.86	1.00	0.43	B×1, C×4
GPT-5.4	off	0.873	1.00	0.86	0.80	0.43	B×2, C×3
GPT-5.4	on	0.870	1.00	0.86	0.80	0.43	A×1, B×2, C×2
GPT-5.1	on	0.846	0.93	0.86	1.00	0.57	C×5
GPT-5.1	off	0.785	0.93	0.71	1.00	0.57	C×5
GPT-5.4-mini	off	0.677	0.86	0.43	0.80	0.14	C×5

GPT系は 5.1（0.785）→ 5.4（0.873）→ 5.5（0.904）と**世代ごとに明確に改善している。** GPT-5.4 までで打ち切っていれば「GPT系はClaude系にやや劣る」という誤った印象を与えるところだった。

ローカル最良（qwen3.6:27b / 0.788）はフロンティア最上位（0.932）の約86%に達し、 GPT-5.4-mini のThinkingオフ（0.677）を上回る。

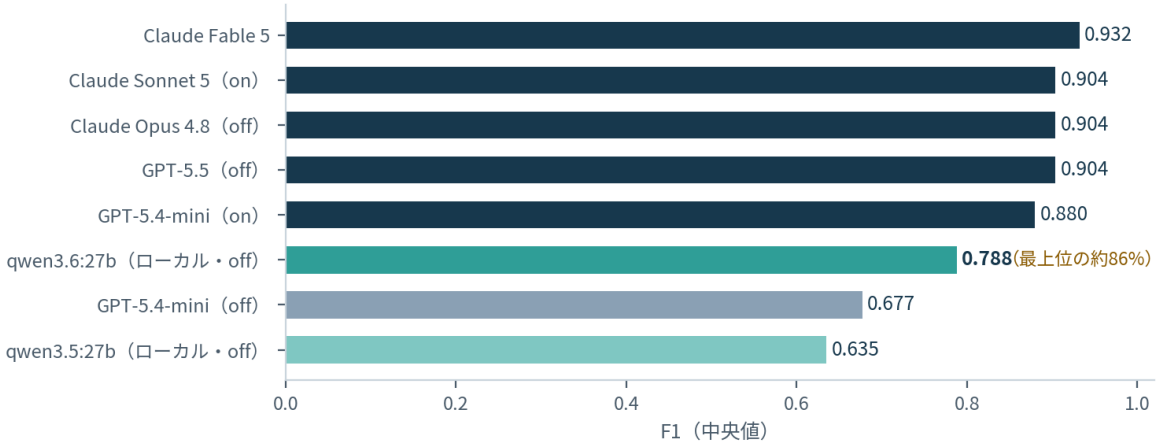

図2　F1中央値の比較 — ローカル最良はフロンティア最上位の約86%

7.3 Thinkingの恩恵はモデルが弱いほど大きい

モデル	Thinkingオフ	Thinkingオン	変化
GPT-5.4-mini	0.677	0.880	**+0.20**
GPT-5.1	0.785	0.846	+0.06
Claude Sonnet 5	0.892	0.904	+0.01
GPT-5.4	0.873	0.870	±0.00
GPT-5.5	**0.904**	**0.892**	**−0.01**
Claude Opus 4.8	0.904	0.889	**−0.02**

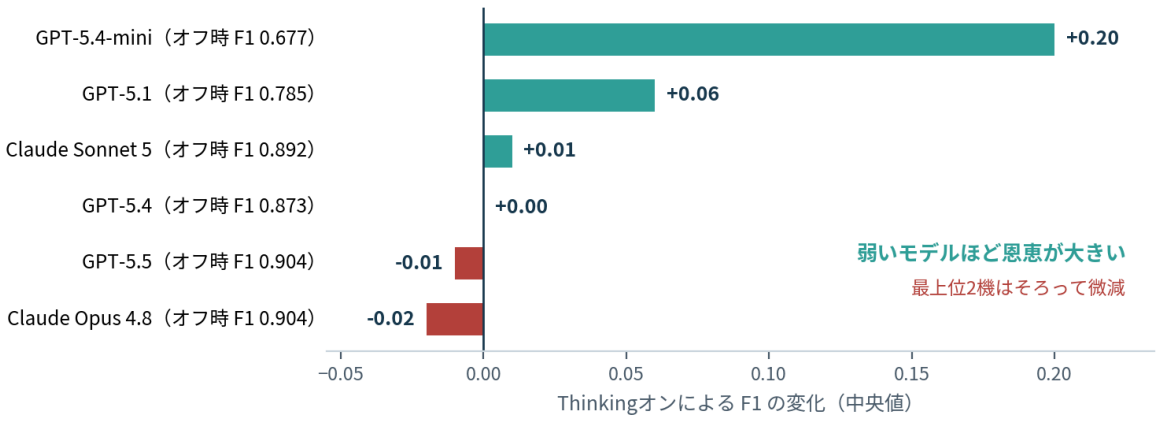

図3　ThinkingオンによるF1変化 — 弱いモデルほど恩恵が大きく、最上位2機は微減

例外なくこの順序になっている。**強いモデルを使えるならThinkingは不要で、弱いモデルほど補正効果が大きい。** GPT-5.4-mini は easy 0.86→1.00、medium 0.43→0.86 と全項目で改善し、事実上「別のモデル」になる。

特筆すべきは、**両社の最新最上位（GPT-5.5 と Claude Opus 4.8）がそろって微減**していることである。 Thinkingが効かなくなる点まで含めて、この傾向が両社で一致している。

これは本稿の皮肉な構図を作る。**Thinkingを最も必要とするのは小さいモデルだが、 ローカルの小さいモデルではそのThinkingがコンテキスト上限で潰れる。**

7.4 小さいモデルほど思考が長い（クラウドでも同じ）

`max_tokens` を16,000から48,000に引き上げた際の生成トークン中央値:

モデル	Thinkingオン時の出力トークン
GPT-5.1	16,315
GPT-5.4-mini	**23,055**

小さく安いモデルの方が4割多く思考トークンを消費している。 ローカル側で観測した「4Bの税率8.73倍 > 27Bの5.70倍」と同じ傾向がクラウドでも起きている。　モデルを小さくしても、思考トークンが増えて時間とコストで返ってくる。

7.5 正解ラベルの妥当性検証

フロンティア55試行の項目別正答率から、gold の欠陥を検出できた。

- `easy` **14項目は 93〜100%** で、下限として正しく機能している
- **D13（負荷試験は専用環境を新設）は11%** しか取れず、独立検証者の「明確に抽出すべき」という　判定が実測で否定された → **goldから許容リストへ移動**（v1.2）
- **複数ステップの計算3項目は 0〜20%** で、最上位モデルでも取れない（§8で詳述）

---

8. 指示の明示性が導出精度を支配する

8.1 自然実験としての発見

`hard_thinking` 7項目を導出の種類で分けると、きれいに割れた（フロンティア45試行時点）。

項目	導出の種類	正答率	プロンプトでの指示
D5	条件分岐の解決	100%	—
A6	相対日付→絶対日付	**96%**	**あり**
A7	相対日付→絶対日付	**89%**	**あり**
D6	消去法	53%	—
N7	複数ステップの計算	20%	なし
N6	複数ステップの計算	2%	なし
N8	複数ステップの計算	**0%**	なし

現行プロンプトにはこう書いてある。

> 「来週の金曜」のような相対的な期限表現は、会議日を基準に YYYY-MM-DD 形式の絶対日付に変換する。

日付の導出は指示したから89〜96%で実行される。算術の導出は指示しなかったから0〜20%。 どちらも「本文に literal に書かれていない値を導出する」という同種の作業であり、 モデルも文書も難易度タグも同じ。違うのは**指示の有無だけ**である。

これは当初「複数ステップの計算はフロンティアでも解けない」と解釈しかけた現象だが、 プロンプトを読み返すと**抽出指示には「計算せよ」とは一度も書いていない。** モデルが「1人月あたり4.5万円」と本文どおり転記するのは、**指示に忠実な振る舞い**とも読める。 つまりこれはモデルの推論能力の限界ではなく、**タスク設計と正解定義の不一致**の可能性が高い。

8.2 対照実験

上記を確かめるため、**差分が1行だけのプロンプトB**を作り、A/B比較を行った。

```
抽出のルール:
  - 検討中・保留のものは decisions に含めない。確定したものだけを含める。
  - 会議中に内容が変更・訂正された項目は、最終的な状態を書く。
  - 「来週の金曜」のような相対的な期限表現は、会議日を基準に YYYY-MM-DD 形式の絶対日付に変換する。
  - 氏名は会議中の表記どおりフルネームで書く。
+ - 重要数値は、最終的な値が本文中にそのまま述べられていない場合でも、本文中の情報から
+   計算で確定できるなら計算した結果を書く。計算できない場合のみ、述べられたとおりに書く。
```

特定の問題を名指しする例（「1.2倍を掛けよ」等）は**意図的に入れていない。** それは答えを教えることになり、測定として無意味になるためである。

8.3 結果：1行で+28〜61pt、副作用なし

フロンティア6モデル×Thinking2条件×5試行で A/B を比較した。

項目	種類	A（指示なし）	B（指示あり）	変化
N7 検証環境サーバ台数（9台）	複数ステップの計算	24%	**85%**	**+61pt**
N6 今期のシステム予算（600万円）	複数ステップの計算	2%	**42%**	**+40pt**
N8 ライセンス費用（108万円）	複数ステップの計算	0%	**28%**	**+28pt**
A6 相対日付（2026-05-22）	日付導出	91%	92%	+1pt
A7 相対日付（2026-05-25）	日付導出	85%	88%	+2pt
D5 条件分岐の解決	推論	96%	100%	+4pt
D6 消去法	推論	55%	57%	+2pt

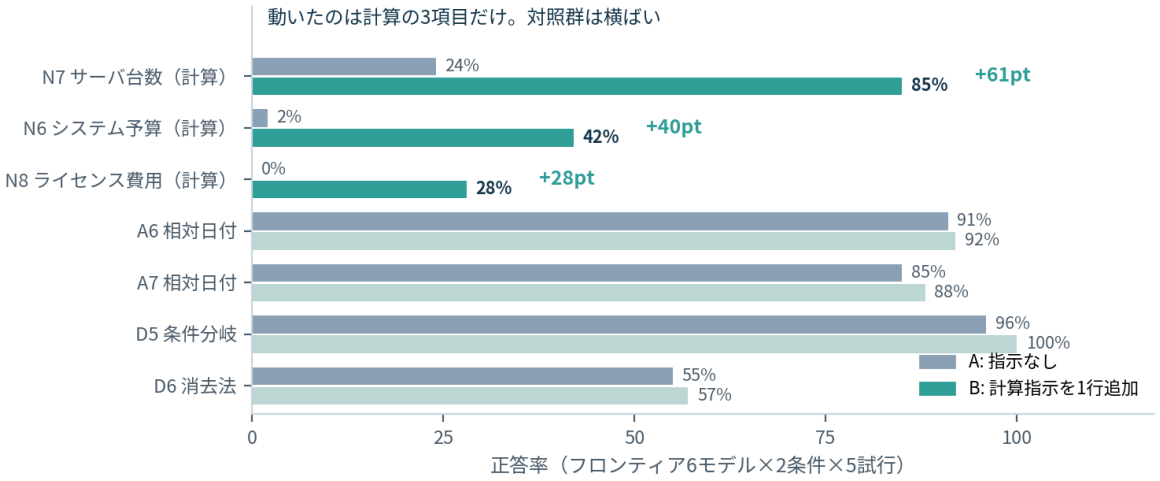

図4　プロンプトA/B比較 — 計算指示1行の追加は計算3項目のみを+28〜61pt動かした

対照群が動いていないことが決定的である。 すでに指示済みだった日付の変換（+1〜2pt）も、指示と無関係な条件分岐・消去法（+2〜4pt）も横ばいである。 **動いたのは指示を追加した複数ステップの計算だけ**（+28〜61pt）。 プロンプトの差分は1行なので、原因は他に求めようがない。

難易度タグ別でも同じ構造が出た。

タグ	A	B	変化
easy	97%	98%	+1pt
medium	82%	84%	+2pt
hard_size	91%	91%	+1pt
hard_thinking	**50%**	**70%**	**+20pt**
hard_implicit	92%	93%	±0pt

副作用は観測されなかった。 指示を1行増やすと他の項目が崩れるという懸念は当たらなかった。

8.4 ただし「解決」はしていない

指示してもなお **N8 は28%、N6 は42%** に留まる。 3項の掛け算（4.5万円/人月 × 8人 × 3ヶ月）は、明示的に指示しても4回に1回強しか正解しない。

> **失敗の主因は指示不足だった。しかし指示すれば解けるわけでもない。**

実務上の教訓は二段構えになる。 **導出値が欲しいなら明示的に指示せよ。ただし算術の正しさは別途検証せよ。**

8.5 ローカルモデルでは効かない

同じA/Bをローカル4モデル（Thinkingオフ）でも実施した。結果は対照的である。

モデル	F1（A→B）	hard_thinking（A→B）	easy（A→B）
qwen3.5:4b	0.431 → 0.618	0.00 → 0.14	0.57 → 0.71
qwen3.5:9b	0.561 → 0.542	0.14 → 0.14	0.71 → 0.71
qwen3.5:27b	0.635 → 0.710	0.29 → 0.29	0.86 → 0.86
qwen3.6:27b	0.788 → 0.758	0.29 → 0.29	0.93 → 0.86

`hard_thinking` はほぼ全く動いていない（フロンティアは 50%→70%）。 F1が上下しているのは他項目の揺らぎであり、方向も一定しない。

qwen3.5:4b だけ F1 が 0.431→0.618 と大きく上がった。 ただし `hard_thinking` は 0.00→0.14 とほとんど動いていないため、指示の効果とは考えにくい。 4Bは元々ばらつきが大きく（タスクAでも F1 0.302〜0.667）、その反映と見るのが妥当である。 いずれにせよ5試行では断定できない。

含意: 指示追従能力そのものがモデルの実力である。 フロンティアは「計算せよ」と言えば計算するが、ローカルモデルは言われても計算できない。 プロンプトエンジニアリングは**能力の底上げではなく、既にある能力を引き出す作業**にすぎない。

> **フロンティアモデルではプロンプト1行が +61pt を生む。同じ1行がローカル27Bでは 0pt である。** プロンプト改善の投資対効果は、使っているモデルの実力に強く依存する。

フロンティアだけを測っていれば「プロンプトを直せば解決する」と一般化するところだった。 **ローカルとフロンティアの両方を測ったからこそ、この限界が見えた。**

8.6 この節が示す方法論上の問題

当初のデータ（タスクAのみ）を見れば、 「複数ステップの計算はフロンティアモデルでも0〜20%しか解けない」と結論するところだった。 これは**測定対象がモデルの推論能力ではなく、こちらのプロンプト設計だった**ことに 気づかなければ、そのまま誤った知見として公開されていた内容である。

気づけたのは偶然だった。 `hard_thinking` タグの中に、 指示した導出（日付）と指示しなかった導出（計算）が両方入っていたためである。 **同じタグ・同じモデル・同じ文書なのに、89〜96% と 0〜20% に割れる。** この説明のつかない差が手がかりになった。

LLMの能力を測っていると思っているとき、実際にはプロンプトを測っていることがある。 これは §4.3（採点系のバグ）と同じ構造の問題であり、 測定系そのものを疑う手続きが評価実験には不可欠である。

---

9. 補論：JSON整形を分離しても品質は変わらない

実務者の経験知として「JSON必須にすると性能が落ちる。まずテキストでまとめさせ、 次にJSON化する2回に分けると性能と書式が安定する」という指摘がある。これを実測した。

設計: 第1段は、1段構えのプロンプトから**JSONスキーマの記述だけを取り除いた**ものである。 第2段は第1段の出力だけを受け取り、形式変換のみを行う（元の会議録は渡さない）。 抽出すべき内容も抽出ルールも同一にし、差分が形式指示の有無だけになるようにした。

結果（qwen3.6:27b、各5試行の中央値）：

Thinking	構え	F1	recall	precision	hard_size	JSON有効
off	1段	**0.776**	0.676	**0.893**	0.80	5/5
off	2段	0.758	**0.676**	0.862	**1.00**	5/5
on	1段	**0.750**	0.649	**0.909**	0.80	5/5
on	2段	0.698	**0.649**	0.846	**1.00**	5/5

recall が完全に一致した（0.676 / 0.649）。取りこぼしの量は1段でも2段でも変わらない。 差がついたのは precision だけで、2段の方がやや余計な項目を出す。 したがって**「形式の制約が読解を阻害する」という仮説は、このモデル・このタスクでは支持されない。**

ただし `hard_size` は2段が明確に優位（0.80 → 1.00、両条件で）。 離れた箇所への参照・訂正を要する項目は、いったん自然言語で整理してからJSON化した方が正確だった。

構造的には狙いどおりだった。 第2段は入力が約20分の1（1,398〜1,496トークン）に縮む。 そのため思考が4,668〜5,405トークン生成しても、**num_ctx 32768 のまま完走する。** §5.4で示した「Thinkingを使うには65,536が必要」という制約を、 2段構えはメモリを増やさず回避できる。本タスクでは品質が伴わなかったため利点にならないが、Thinkingが有効なタスクでは意味を持つ可能性がある。

代償は時間で、2段構えは1試行あたり約287秒（第1段110秒＋第2段177秒）。 1段構えのThinkingオフ（約156秒）の**1.8倍**である。

検証の限界: qwen3.6:27b はもともとJSON出力が安定している（両構成とも5/5）。 つまり経験知の中心である「書式が安定する」効果を検証できる条件になっていない。 JSONを壊しやすいモデルで測る方が適切である （本実験では qwen3.5:27b がThinkingオフでも5回に1回壊れた）。 また1モデル・1タスクの結果であり、一般化はできない。

---

10. 実務判断のための指針

10.1 議事録生成（社内会議・約6時間相当の記録、23.4kトークン）

モデル	1本あたり所要	F1	グレード	判断
qwen3.5:4b	25秒	0.431	C	**下書きにも使えない**（easy 0.57）
qwen3.5:9b	47秒	0.561	C	実務投入不可
qwen3.5:27b	2分32秒	0.635	C	Thinkingオフでも5回に1回JSONが壊れる
qwen3.6:27b	**2分37秒**	**0.788**	C（B×1）	**唯一の候補。ただし人手レビュー必須**

10.2 運用上の禁止事項

- **num_ctx 32768 でThinkingを有効にしてはいけない。** qwen3.5系では時間が6〜9倍になった上で 出力が得られない。qwen3.6:27b でも上限まで残り1,000〜1,800トークンしかなく危険域である
- **カタログ値や短文での tok/s を長文脈の見積もりに使ってはいけない。** 1〜2割落ちる

- **導出値が欲しいなら明示的に指示する。** 「抽出して」では計算されない（§8）

10.3 ローカルかクラウドか

	F1	備考
ローカル最良（qwen3.6:27b）	0.788	完全ローカル、2分37秒/本
GPT-5.4-mini（Thinkingオフ）	0.677	**ローカルが上回る**
GPT-5.4-mini（Thinkingオン）	0.880	
Claude Sonnet 5	0.892〜0.904	
Claude Opus 4.8	0.889〜0.904	**唯一グレードAを安定して出す**
Claude Fable 5	0.932	最高値

機密性の要求が高く、品質0.79で許容できるなら、ローカル27Bは現実的な選択肢である。 一方、自動連携（グレードA）が必要ならフロンティア上位以外に選択肢はない。

---

11. 限界

1. `hard_size` **は5項目しかない。** 統計的な結論ではなく傾向の示唆に留まる
2. **1会議・1ドメインのみ。** 社内IT刷新プロジェクト会議であり、ドメイン一般化の主張はできない
3. **営業会議・商談は含まない。** SFA連携用途の評価には使えない（次回スコープ）
4. **難易度タグは設計者の想定**であり、実測前の仮説にすぎない
5. **プロンプトキャッシュの影響**: 同一プロンプトを繰り返すと初回のみプロンプト処理コストを含む （4B: 71.2s → 33.0s → 24.7s → 15.4s）。実運用は毎回異なる会議録を処理するため、 総時間は初回の値に近い。本稿の時間は中央値であり、この点を割り引いて解釈する必要がある
6. **フェーズ4は量子化方式が異なる**ため「同じモデルの比較」ではない
7. **Ollamaのバックエンド判定はログからの読み取り**であり、公式ドキュメントの裏付けではない
8. **70Bは検証していない**（次回）

---

12. 再現方法

```
experiment/
  dataset/   schema.json / gold.json / transcript.txt / dataset_notes.md
  scripts/   run_thinking_bench.py / score.py / summarize.py
             run_frontier_eval.py / run_mlx_compare.py
  results/   *.csv / summary.md
  logs/      全実行の生ログ
  env.txt    環境情報
  notes.md   実施中の全判断・逸脱・想定外の記録
```

```
python3 scripts/run_thinking_bench.py --task extract          # フェーズ2
python3 scripts/score.py --logs logs/thinking_bench_extract   # フェーズ3
python3 scripts/run_frontier_eval.py --list-models            # モデルID実在確認
python3 scripts/run_mlx_compare.py --check                    # バックエンド調査
python3 scripts/summarize.py                                  # 集計
```

測定スクリプトはすべて Python 3.9 互換・標準ライブラリのみで書いている。

誠実性のルール: 数値の捏造・補完は行っていない。測れなかったものは「測れなかった」と記録し、 打ち切りは打ち切りとして記録している。実施中の全判断・逸脱・想定外は `notes.md` に残した。

---

13. 次回に向けて

- **70Bクラスの検証**
- **SFA連携用データセット**: 本稿のスキーマは議事録向けであり、商談・営業会議には形が合わない。 商談フェーズ、確度の変化、受注見込み金額、キーパーソン、失注リスクといった項目が要る。 それに対応する営業会議のシナリオも別途必要になる
- **MLX と llama.cpp の速度比較**: 本稿では測定できなかった。 MLXモデル（qwen3.5:27b 相当の4bit/5bit）は取得済みだが、読み込めなかった。 macOS 同梱の Python 3.9 では `mlx-lm` が 0.29.1 までしか入らず、 qwen3.5 アーキテクチャに対応していないためである（`Model type qwen3_5 not supported.`）。 測定には Python 3.10+ の別途導入が要る。 なお調査の過程で、本機の Ollama 0.30.10 が **MLXではなく llama.cpp/GGML の Metal バックエンドで動いている**ことを確認した。 サーバログ上、`mlx` への言及は0件、`ggml` / `llama.cpp` への言及は633件である。 「Ollama は Apple Silicon で内部的に MLX を使う」という理解は、少なくとも本構成では成立しない
- **KVキャッシュ量子化の品質影響**
- **複数ドメイン・複数会議への一般化**

Citation

```
@techreport{ohashi2026localminutes07,
  title       = {ローカルLLMで議事録は作れるか［2026年7月版］：64GB Mac mini・27Bモデルの実用性評価},
  author      = {大橋 功},
```

```
  institution = {株式会社喋ラボ},
  year        = {2026},
  url         = {https://shabelab.com}
}
```