ChatGPT

TOON in Hebbia’s Financial AI Workflows: Benefits and Trade-offs

Hebbia’s Data Schemas – Which Suit TOON vs. JSON?

Matrix Grid Rows (Structured Tables): Hebbia’s **Matrix** often outputs large tables – “thousands of rows and dozens of columns of LLM-generated content in a single grid” [1]. These tables are uniform and shallow: each row has the same fields (e.g. company, year, metric values). For example, a hedge fund query might produce a table of revenue by year for multiple companies, with every data point linked to a source [2]. Such **regular, tabular data** is ideal for TOON’s compact syntax. JSON would repeat quotes and keys for each row, whereas TOON can represent rows as terse typed tuples (e.g. `Record(company: AlphaCorp, year:2020, revenue:1234)`). This could significantly cut tokens for large tables, which are common in Matrix outputs [2]. In short, **uniform arrays of records (Matrix grid rows)** map well to TOON’s token-efficient format.

ISD Citations (Excerpt + Source References): Hebbia’s **Iterative Source Decomposition (ISD)** produces *citation-linked outputs* – answers or facts with pointers to original documents [2]. For instance, Matrix might return a bullet point summary with each sentence annotated to its SEC filing or transcript source. Under the hood, an ISD citation could be represented as a small object: `{ text: “Profit margin was 24%”, source: Doc123 p.5 }`. These are moderately structured (text + metadata). TOON could encode them as `Citation(doc: Doc123, page:5, text: Profit margin was 24%)`. This removes JSON quotes around keys and braces, saving tokens. However, the **payload text** still contains natural language (which needs quotes or some delimiter in TOON). If citations are short and numerous, TOON yields benefits by compressing repetitive key names (“doc”, “page”, etc.). But if the **quoted content** is long, the savings are smaller – we’d still pay for the text tokens. **Bottom line:** ISD citation entries are semi-structured; TOON helps a bit (for the metadata), but the actual quote text remains the bulk. TOON is beneficial if we have many small citation objects, but less so if citations contain long narrative text where JSON overhead is negligible.

Agent 2.0 Context Traces (Nested Decision Logs): Hebbia Agent 2.0 coordinates an *orchestrator* and many sub-agents, producing a structured trace of each step (tools invoked, intermediate results, etc.) [3] [4]. These context traces are rich and hierarchical. For example, an orchestrator’s log might contain an array of sub-agent invocations, each with its own inputs and outputs, which in turn may include chunks of retrieved text or generated summaries. Such **deeply nested, heterogeneous data** is *not* TOON-friendly. Every step can have a different shape (one might have a list of document IDs, another a blob of text). JSON’s flexibility is useful here – it can represent varied structures without a rigid schema. Converting a full agent trace to TOON could become unwieldy, and any token savings from removing braces/quotes might be offset by the complexity of parsing a deeply nested custom notation. Moreover, these traces are primarily for internal logging/observability (often not included verbatim in model prompts) [5]. **Recommendation:** Keep complex, nested agent traces in JSON (or not feed them to the model at all); TOON provides little benefit on verbose, irregular debug data.

Token Savings vs. Parser Overhead in Real-Time Flows

TOON's Efficiency Gains: By stripping away JSON's syntactic cruft, TOON can reduce token count by roughly **30–50%** in structured outputs [6]. Fewer tokens directly mean lower OpenAI API costs and often faster model inference [7]. In Hebbia's context, where some queries aggregate *lots of structured data*, the token savings could be substantial. For example, a Matrix comp table with 50 rows might be, say, ~500 tokens in TOON vs ~800 in JSON – a ~37% reduction. Over hundreds of such queries, this adds up to significant cost and latency improvements. Notably, Hebbia emphasizes scale: their platform processes *billions of tokens per day* across research agents [8]. Cutting 30–60% of tokens in structured parts of those requests would meaningfully alleviate context window pressure and throughput load.

Parser Overhead and Latency: The flip side is that TOON is a custom format – clients (or Hebbia's frontend) must parse and generate it. JSON has ubiquitous, optimized parsers; TOON would require a new parsing step in Python/TypeScript. For real-time user interactions (e.g. a LevFin banker using Matrix live in a meeting), an extra 50ms of parsing can matter. However, the overhead of a well-written TOON parser (essentially a lightweight tokenizer for parentheses and commas) is likely **small** relative to LLM inference time. For instance, an LLM answering a complex query might take 2–5 seconds on the model side, whereas parsing a few KB of output with simple string operations might take only a few milliseconds. In high-frequency automated flows, though, every millisecond counts – e.g. if an agent is making thousands of lightweight calls, cumulative parser cost could add up. Hebbia's system already handles massive parallel requests via Maximizer [9], so any per-request overhead should be scrutinized.

Real-Time Flow Considerations: In **live modeling or document querying** sessions, user experience is king. A **leveraged finance** user might ask Matrix to pull covenant data into a live spreadsheet – they expect near-instant cell updates. Here, an extra client-side parse could introduce slight lag. If the structured answer is small (a few values), JSON's overhead is minimal anyway (maybe tens of tokens), so TOON isn't worth the complexity. Additionally, **reliability** is a factor: JSON outputs can be parsed strictly, and many tools exist to auto-correct minor JSON syntax errors from LLMs. A custom TOON parser might need to handle edge cases (e.g. missing parentheses) gracefully, or the whole answer could fail to parse. This risk is non-trivial in real-time flows where there's no time for manual fixes. In summary, **TOON's token savings translate to speed mostly for large payloads** – in batch or heavy analytic queries, reducing 30% tokens could shave noticeable time. But in single quick Q&A exchanges (e.g. a PE associate querying one document), the savings (maybe a few dozen tokens) won't offset the cognitive overhead of introducing a new format.

Trade-off in Practice: For hedge fund researchers running multi-document analysis, TOON could speed up complex queries by cutting context size (which also lowers model latency) [6]. For a banker quickly querying a single 10-Q, JSON is fine – the response is mostly natural language with perhaps one or two numbers, so token count isn't an issue. Also, Hebbia's **Agent 2.0** already optimizes context by focusing only on relevant info (needle-in-haystack filtering) [10]. This means often *only a small subset* of data is fed to the model, mitigating JSON's usual bloat. Thus, the *relative* benefit of TOON is case-dependent: **big structured outputs = big win; small ad-hoc answers = negligible gain.**

Token Usage Patterns by Team and Data Type

Hebbia's platform supports various teams (finance, law, corporate), each with different data footprints. We break down **input token composition** for three core use-cases and estimate what portion could benefit from TOON:

- **Hedge Funds (Research-Heavy Queries):** Hedge fund analysts load **hundreds of documents** (filings, transcripts, etc.) into Matrix and ask broad questions [11] [12]. A single query might draw on many snippets across years of 10-Ks and earnings calls. Here, **80–90% of the input tokens** tend to be raw text chunks from documents (natural language). The rest might be **structured summaries** or tables the agent generates to organize answers (e.g. a timeline of metrics, a comparison table) [2] [13]. Those structured parts – perhaps ~10–20% of the total context – are **TOON-eligible**. For example, if an output table of key financials spans 5 companies × 5 years, that tabular data could be compacted with TOON, saving ~50% of those tokens. Overall, however, the *bulk of hedge fund queries' context* is unstructured text (which TOON can't compress much). So maybe ~15% of the total tokens in a typical hedge fund QA context could be trimmed via TOON. Still, given the scale (these teams literally query *decades of filings in one go* [14]), even a 15% reduction on multi-thousand-token contexts is meaningful.

- **Leveraged Finance (Investment Banking) – Live Modeling:** LevFin teams use Hebbia to **populate and analyze financial models** and to pull data for pitch decks and memos [15]. Their interactions often mix **numeric data** (financial metrics, deal terms) and **textual analysis** (qualitative descriptions from CIMs or credit agreements). We estimate their input contexts might be more balanced: ~50% structured data (e.g. tables of financial ratios, lists of covenants) and 50% unstructured (company descriptions, risk factor text). This is because bankers frequently ask the AI to **extract data points** (which become numbers in a spreadsheet) and also to **explain or summarize** (which is narrative). Structured parts here are things like a debt schedule, a list of EBITDA figures by year, etc., which are excellent candidates for TOON encoding (uniform lists of figures). In LevFin workflows, **TOON-eligible content could be ~30–50%** of the context by tokens – significantly higher than in pure research use. If a banker asks, "List EBITDA, interest expense, and coverage ratios for the last 5 years for Company X," the output is essentially a small table of numbers (ideally compressed via TOON). Meanwhile, the accompanying explanations ("these figures show improving coverage…") remain plain text. This means **TOON could notably cut token usage** for bankers' data-heavy queries, speeding up responses without much downside – bankers value quick turnarounds in live deal discussions.

- **Private Equity (Due Diligence & Research):** PE teams straddle both worlds: they do deep research on data room documents (like hedge funds) and also extract structured deal data and KPIs (like bankers). In a diligence Q&A, ~70% of tokens might come from documents (e.g. parsing legal contracts, technical reports) and ~30% from structured outputs (like checklists, financial summaries). PE use-cases often involve *iteratively querying many docs and building an internal report*. For instance, they might generate a list of all contracts with change-of-control clauses – that list is structured (contract name, clause summary, source). They then might also ask for a narrative risk summary. So we'd peg **~30% of PE context tokens as TOON-eligible** on average – higher during the data extraction phase, lower during narrative synthesis. Over an entire diligence process (which could involve thousands of pages analyzed), using TOON for the structured interim results can trim a sizeable chunk of tokens (and thus cost). Notably, Hebbia claims PE firms save 20–30 hours per deal by using the AI for screening and analysis [16] – a sign that the AI is reading and outputting *huge volumes* of information that humans otherwise would, and a portion of that is tabular/structured.

Total Context Volume & TOON Eligibility: Across Hebbia's financial user base, we can extrapolate: Most heavy workflows (research, diligence) are text-dominated, but a **meaningful minority (~20% of context tokens)** are in structured formats (tables, lists, key-value outputs). Those are the tokens TOON can target. Considering Hebbia processes *over 250 billion tokens/month* [9], even converting 20% of that to a format that's 50% more efficient could shave ~25 billion tokens off monthly usage. In practice, not all of those tokens are currently sent to models (some may be post-processed client-side, etc.), and TOON wouldn't apply to tokens that are part of the model's own vocabulary for raw text. But it gives a sense of scale: **TOON could realistically reduce total token consumption by high single-digit percentages** platform-wide, which at Hebbia's scale is nontrivial. The key is that it's concentrated in certain flows (e.g. big Excel-like outputs, long lists of results). In those scenarios, TOON might save on the order of 30–60% of the tokens for that particular query [6] – for example, a large table result – while across all queries the average savings is lower. We should also note that **not all context is TOON-convertible** – large passages of text (the majority of content) remain untouched. So the focus should be on those high-impact structured pieces within the context.

How Data Flows Through Hebbia (Agents, Matrix, and Citations)

Figure: Hebbia's Agent 2.0 orchestrating a Matrix query. **Left:** The Orchestrator agent delegates tasks to specialized sub-agents (Reading, Retrieval, Output, etc.) rather than handling everything in one prompt [17] [18]. **Center:** A *Read Matrix* sub-agent selects relevant cells/columns from the Matrix (spreadsheet) via a tool, instead of dumping the entire grid into the prompt [19]. This targeted context extraction keeps token usage efficient. **Right:** An *Output* agent composes the final answer, possibly calling a formatting tool to assemble results for the user. TOON could be applied in the communication between these agents – for example, the Read Matrix agent could return data as a compact TOON structure to the Output agent, minimizing tokens passed along. (Currently, the system uses text-based objectives and structured context objects [3] [20] – introducing TOON here would mean those context objects are serialized in a token-optimized way for the LLM.) The figure illustrates how data flows from user query to orchestrator, then to sub-agents and back, with each agent focusing on its piece. *This modular "synchronized yoyo" approach* [17] *ensures that only necessary data (like the chosen Matrix columns) enters the LLM's context, reducing noise and cost.*

Figure: Multi-hop Output Agent writing a cited report (conceptual). **Step 1 (Hop 1):** The Output agent writes the first section of a report (e.g. *"Tesla – Financial Analysis"*), pulling in facts from sources (Tesla's filings for 2023, 2024) and citing each fact [13]. Hebbia's ISD ensures every claim includes a reference to the precise document snippet [21]. **Step 2 (Hop 2):** The agent then proceeds to the next section (e.g. *"GM – Financial Analysis"*), again retrieving relevant data and citations. Throughout these hops, the agent **cannot fit all raw data at once** due to context size, so it works iteratively – writing a section, then using that output (or a distilled summary of it) as context for the next section [22]. The figure shows how citations (right side, e.g. specific revenue figures with footnotes) flow into the written report. This *audit-trail by design* is a hallmark of Hebbia's system – by the end, the compiled report is "fully cited, exhaustive" [22], and each data point can be traced to its source. **Where could TOON help here?** Potentially in how the agent internally represents the list of citations or interim structured data between hops. For instance, before writing, the Output agent might gather a list of `(Fact, Source)` pairs. Representing that list in TOON (as `Fact(source: X, value: Y)`) could save tokens versus a verbose JSON list, allowing more facts to fit in the context window.

However, the final **user-facing output** is natural language with inline citations (which is already concise, e.g. "[Q1 2024 revenue grew 10% [2]]"). The citations themselves are often just numeric indices or short tags in the text, which are minimal token-wise. So TOON's main utility is in the behind-the-scenes data passing (ensuring the agent has as many facts on hand as possible without overrunning context). Hebbia's design of ISD is model-agnostic and emphasizes precise linking [21] – a token-efficient format like TOON could further improve the "infinite context" illusion by packing more evidence into each agent step.

Recommendations: Where to Use TOON vs. JSON in Hebbia

Use TOON for Uniform, Tabular Data: Whenever Hebbia's agents produce **regular grids, lists, or arrays of entries**, prefer TOON. This includes Matrix outputs like financial statements, comp tables, query results that naturally form a table, or lists of entities with attributes. These schemas are repetitive and shallow, so TOON yields 30–60% token reduction with no loss of information [6]. For example, a "matrix" of companies and metrics should be serialized in a compact notation (no JSON quotes or braces). This will **cut cost and latency** for high-volume financial modeling tasks where dozens of numeric columns are extracted [1]. TOON is also ideal for any **structured API calls** within the agent framework – e.g. if the Orchestrator sends sub-agents a set of parameters or if the Output agent returns a structured answer to the UI, those can be token-optimized. Essentially, use TOON as the "lingua franca" for any internal data exchange that looks like a table or a list of objects.

Use TOON for High-Volume Metadata and Small Objects: Some flows involve lots of small JSON objects, where keys/key names contribute disproportionate overhead. A prime example is a large set of **citation objects** or search results. If Hebbia ever feeds the model something like a list of document snippets with identifiers, TOON can trim the fat. E.g., instead of `[{ "doc": "ABC", "page": 5}, {...}]` × 50, you could have `(doc:ABC, page:5), (...)` in TOON – a significant savings, allowing more snippets per context. In general, **header info and metadata** (IDs, timestamps, etc.) are great candidates for TOON encoding, since they are highly structured and frequent. Hebbia's agents tag each message with the producing agent and turn IDs for tracing [5]; if such info were ever included in prompts or logs to the model, it should be in a minimal form (or omitted entirely). A hybrid approach could be used: **TOON for the structural "skeleton," JSON for any free text "meat."** For instance, an agent could output: `Answer(section: Overview, data: [Point(text: "Sales up 10%", source: 123), Point(text: "Margin up 2%", source: 456)])`. Here `section` and the list structure are in TOON, while each point's actual text remains a normal string. This way, we remove most quotes except those absolutely needed around arbitrary text. Such hybrid formatting leverages TOON's efficiency for repetitive wrapper structure without trying to cram long natural language into a custom syntax.

Avoid TOON for Deeply Nested or Irregular Data: If data is highly nested or each item has a different shape (which is rare in Hebbia's end-user outputs, but common in internal traces), JSON (or another flexible encoding) is safer. Agent 2.0 contexts, as discussed, should stay in their native data structures on the back-end – there's no need to inject the full execution trace into model input. But if there ever is a need to ask an LLM to analyze a complex data structure (e.g. debug info or a rich analytic object), it might be better to **serialize to JSON or a well-known format**. LLMs have been trained on JSON patterns and might handle them more predictably in weird edge cases. TOON is not magic – feeding a convoluted tree of objects in TOON won't help the model unless the content itself is relevant. In short, use TOON **selectively for the parts of the context that are structured and important**, not for everything by default.

Avoid TOON when Latency/Robustness Trumps Token Cost: In some real-time user interactions, the priority is to respond quickly and reliably. If a context or response is relatively small (say <100 tokens as JSON), the absolute savings from TOON might be under a few dozen tokens – worth only a few milliseconds of model time. Introducing TOON there adds complexity (a custom parser, potential formatting errors) for negligible gain. For example, a quick question answering "What's the P/E ratio of X?" might yield a one-liner answer and a numeric value – JSON overhead is trivial. It's wiser to return a simple JSON or even plain text. Additionally, TOON is new; models aren't natively trained on it. While it's designed to be LLM-friendly, we should ensure our prompts include instructions or examples so the model knows to output TOON correctly. If not, the model might accidentally hallucinate a brace or misplace a comma, and our parser could choke. In high-stakes settings (e.g. a demo to a client, or when integrating with external systems expecting JSON), a malformed TOON output could be worse than a slightly longer JSON output. **Thus, avoid TOON in latency-critical or externally facing APIs until it's well-tested.**

Hybrid Strategies – Marry TOON with JSON: Hebbia can adopt a pragmatic middle ground: use **TOON for what it's best at, JSON for the rest**. One approach is indeed using **TOON for keys/labels and JSON for values**. For instance, an output could be a JSON object where one field contains a TOON-encoded array. Conversely, a top-level TOON structure could carry mostly numbers/IDs, and any free-form text fields are simply quoted (since TOON syntax would treat them as strings anyway). Another idea is to let the LLM use TOON for intermediate reasoning (where we care about token count), but then have the final answer converted to JSON for the client app. Since Hebbia's agents already do multiple-step processing (with an Output agent assembling the final answer) [22], that Output agent could take a TOON-serialized interim result and **transform it into clean JSON** before returning to the user – giving us the best of both worlds (efficient model usage, standard output format). For example, the agent could internally get `DealTerms(company: X, leverage:5.0, covenant: none)` and then output to the user as a nice JSON `{"company":"X","leverage":5.0,"covenant":"none"}` or even a sentence with citation. The user doesn't see TOON, but we reaped its benefits behind the scenes.

📊 **Benchmark and Iterate:** We recommend Hebbia conduct small-scale benchmarks on real-world workflows. For a given heavy query (say, "Build a table of all tech giants' profitability" which triggers multi-document retrieval and a large table), run it with standard JSON vs. with TOON-format prompts, and measure end-to-end latency and token count. This will concretely quantify the trade-offs. It's important to ground any TOON adoption in realistic token cost analysis (the question prompt rightly warns not to exaggerate savings). If, for instance, only 10% of Hebbia's workload is highly structured, then focus TOON on that 10% rather than refactoring everything. The goal is to **reduce the total cost per query for key high-volume clients (hedge funds, PE, etc.) without compromising reliability**. Given Hebbia's scale, even a single-digit percent reduction in tokens can translate to huge dollar savings monthly, as well as the ability to push more data through the "infinite context" pipeline.

Conclusion: A Forward-Looking Assessment

TOON offers a compelling efficiency boost for Hebbia's AI workflows, but it's not a silver bullet for all data. It aligns best with Hebbia's **structured, repetitive outputs** – an area that will only grow as users demand more comprehensive analyses (think larger comp tables, longer timelines, bigger "AI-built" models). In those areas, TOON could become the default format, cutting 30–60% of tokens and enabling the system to handle even more documents or deeper analysis within the same context limits [6]. For unstructured data and ad-hoc responses, Hebbia should stick to the proven JSON/text approach to maintain speed and simplicity.

Crucially, adopting TOON should *not* disrupt user workflows. Founders like Lucas and George can introduce it under the hood – speeding up the engine without changing the car's interface. Over time, as both LLMs and users become accustomed to more structured interactions, TOON could even open up new capabilities (e.g. allowing users to request outputs in a structured form directly for integration with Excel or BI tools). But that's looking ahead. In the near term, the strategy is: **use TOON where it clearly helps (uniform data), avoid it where it adds risk (complex or time-critical scenarios), and consider hybrid formats to get the best of both.** By following these guidelines – and continuously validating with real usage data – Hebbia can leverage TOON to trim fat from their context without cutting into the muscle of their user experience.

Sources:

- Hebbia Blog – *Divide and Conquer: Multi-Agent Redesign* (Lucas Haarmann et al., June 2025) [1] [19]
- Hebbia Blog – *Inside Hebbia's "Deeper" Research Agent* (William Luer et al., July 2025) [21] [8]
- Hebbia Blog – *How Hedge Funds Use Hebbia* (Sonal Gupta, Nov 2025) [2] [13]
- OpenAI Case Study – *Hebbia automates 90% of finance and legal work* (Oct 2025) [23] [15]
- LinkedIn Article – *TOON vs JSON: Token-Oriented Notation Future for LLMs* (Pushpendra N., Nov 2025) [6] [24]
- Hebbia Blog – *Maximizer: High-Scale LLM Scheduling* (Mar 2025) [9]

---

[1] [3] [4] [5] [10] [17] [18] [19] [20] Divide and Conquer: Hebbia's Multi-Agent Redesign
https://www.hebbia.com/blog/divide-and-conquer-hebbias-multi-agent-redesign

[2] [11] [12] [13] [14] How Hedge Funds Use Hebbia
https://www.hebbia.com/blog/how-hedge-funds-use-hebbia

[6] [7] [24] TOON vs JSON: Why Token-Oriented Object Notation is the Future for LLMs
https://www.linkedin.com/pulse/toon-vs-json-why-token-oriented-object-notation-future-neniwal-sdn3c?trk=public_post

[8] [21] [22] Inside Hebbia's "Deeper" Research Agent
https://www.hebbia.com/blog/inside-hebbias-deeper-research-agent?utm_source=www.theneurondaily.com&utm_medium=referral&utm_campaign=is-meta-s-personal-superintelligence-push-going-to-kill-facebook

[9] Maximizer: Hebbia's Distributed System for High-Scale LLM Request Scheduling
https://www.hebbia.com/blog/maximizer-hebbias-distributed-system-for-high-scale-llm-request-scheduling

[15] [16] [23] Automating 90% of finance and legal work with agents | OpenAI
https://openai.com/index/hebbia/