

PRODUCT EXPLANATION & VISUAL WALKTHROUGH

AI Token Compressor

Prepare the evidence.
Reduce the input.
Show what changed.

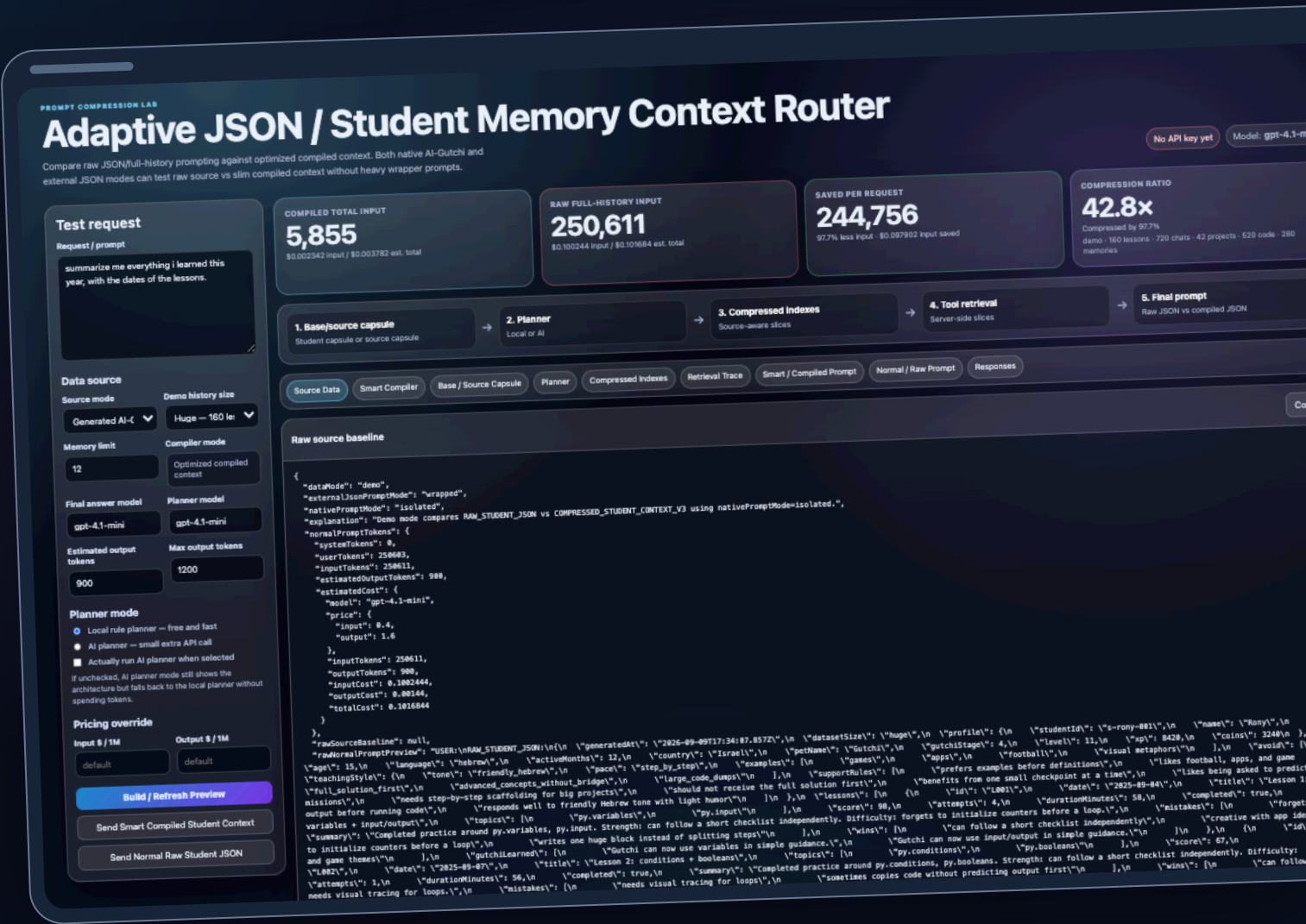
ORGANIZE

CALCULATE

SELECT

97.7%

Less estimated input
in one synthetic local test.
See the full context on page 12.



A guided demo. No login required.

AI Token Compressor prepares large amounts of structured information before an AI receives a question. This document shows the product, its decisions and the evidence behind the result.

03 **The idea**
Why prepare data before asking an AI? [↗](#)

04 **The workspace**
A guided tour of the actual interface. [↗](#)

07 **Demo 01 · Student history**
Follow the selection and inspect the reduction. [↗](#)

13 **Demo 02 · Custom data**
Upload, interpret and calculate from JSON. [↗](#)

16 **AI connection & costs**
What works locally; what a live call adds. [↗](#)

19 **What was built**
The complete application and its boundaries. [↗](#)

22 **Present this demo**
An eight-minute walkthrough, ready to use. [↗](#)

24 **Evidence & verification**
Sources, local checks and screenshot notes. [↗](#)

A note on the name The screenshots use the internal title “Adaptive JSON / Student Memory Context Router”. Throughout this document, the product is called AI Token Compressor.

The question is small. The source is not.

An AI may need an answer about a large history without needing every original record in its original format.

A SIMPLE EXAMPLE

"How has this student progressed over the year?"

THE FULL-HISTORY APPROACH

Send the entire file again.

Lessons, dates and repeated labels

RAW RECORDS

Chats, projects and code submissions

MORE HISTORY

Notes, assessments and formatting

MORE TEXT

The question travels with the original full-length source, whether every detail is useful or not.

THE PREPARED-CONTEXT APPROACH

Build a smaller evidence package.

A compact profile and skill picture

ORGANIZE

Relevant timelines and checkpoints

SELECT

Supported figures calculated in advance

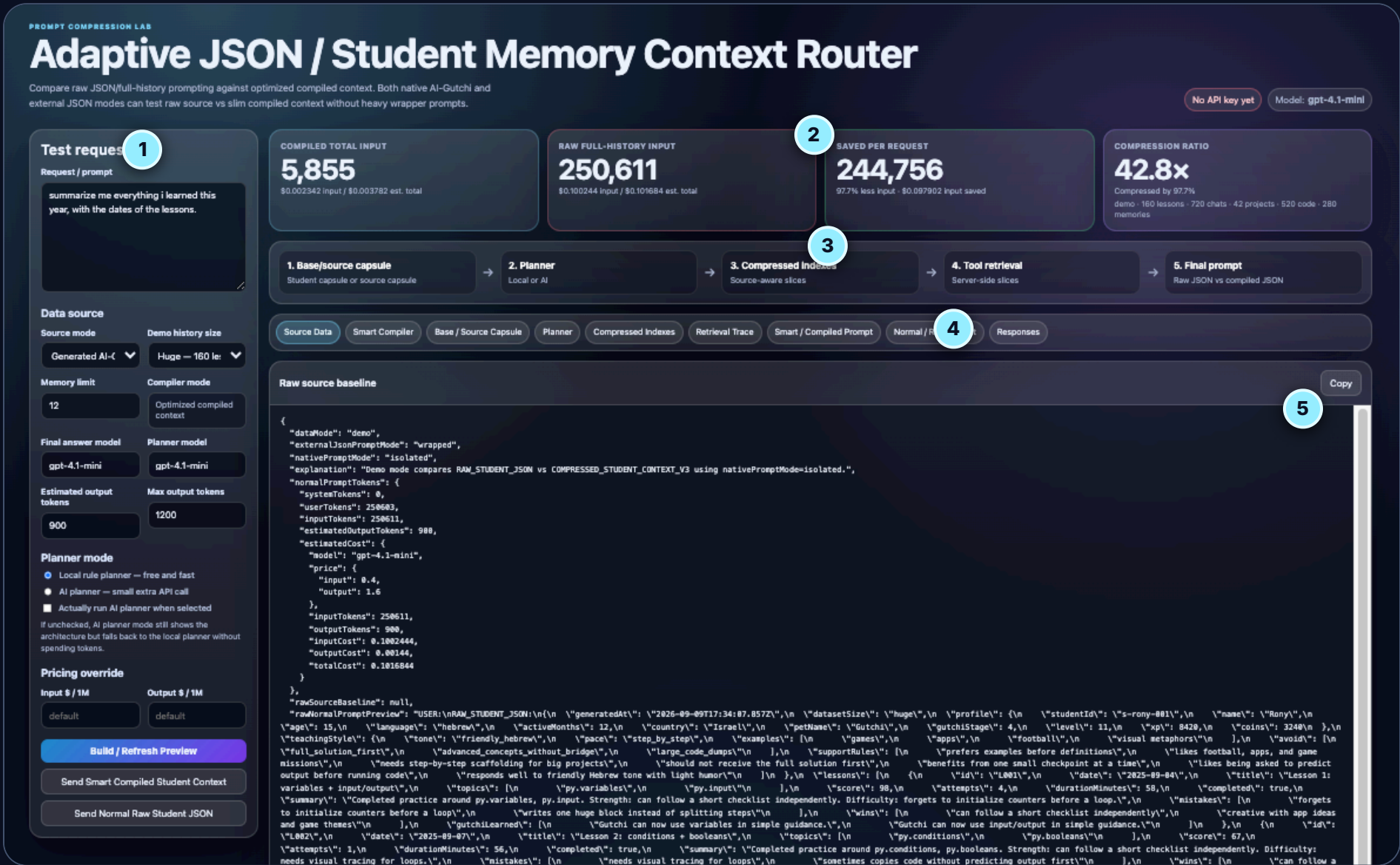
COMPUTE

The model receives prepared context and can use that evidence to formulate its answer.

This is selective preparation, not a promise that every detail survives. Reviewing what was included is part of the workflow.

Meet the workspace.

One screen connects the question, the source, the preparation steps and the comparison.

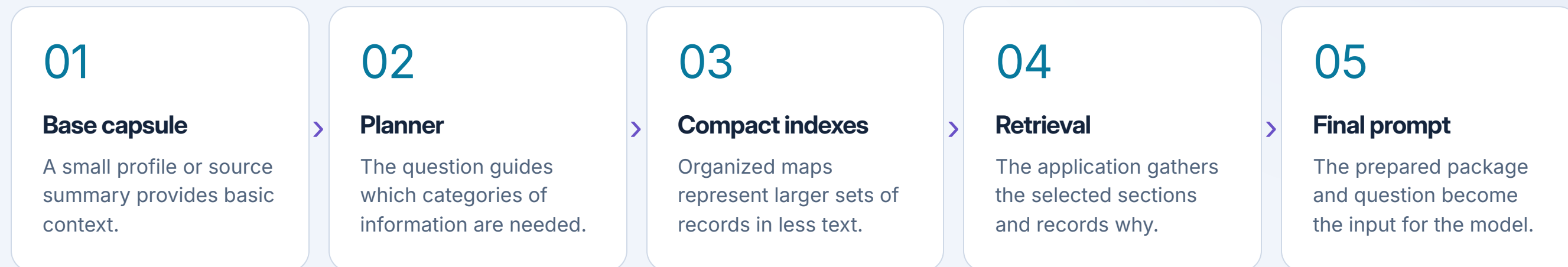


Supplied product screenshot. The “No API key yet” status identifies a local preview, not a live AI answer.

- Request controls**
Choose the question, data source and how much detail to allow.
- The comparison**
Read input sizes, estimated cost, savings and the compression ratio.
- The pipeline**
Follow the visible stages that assemble the prepared request.
- Inspection tabs**
Open the source, plan, selected sections and final prompts.
- The detail panel**
Read the evidence behind the numbers instead of relying on a black box.

Prepare first. Ask second.

The application first organizes the source and builds supported summaries. The dashboard exposes five parts of the prepared request.



The five-stage pipeline as it appears in the supplied application.

NO AI CALL REQUIRED

Local preparation

Formatting, indexing, supported calculations and rule-based selection can run without an AI API key.

WHEN A SERVICE IS CONNECTED

Optional AI involvement

A connected AI can help choose the sections or generate an answer. Those are additional, explicit calls.

Start with a question and a source.

The left-hand panel sets up the comparison. “Build / Refresh Preview” prepares the inputs before an answer is requested.

Test request

Request / prompt

summarize me everything i learned this year, with the dates of the lessons.

Data source

Source mode

Generated AI-Gl ▾

Demo history size

Huge — 160 less ▾

Memory limit

12

Compiler mode

Optimized compiled context

Final answer model

opt-4.1-mini

Planner model

opt-4.1-mini

1

Write the question in everyday language

Ask for a summary, help with a coding difficulty, or a supported calculation. The wording influences the selection.

2

Choose the source

Use generated student history, a built-in JSON example, or your own valid JSON. Generated histories offer 38, 96 or 160 lessons.

3

Build a preview

Inspect the raw and prepared inputs, the selected information and the estimated usage. A local preview is not a generated AI answer.

Memory limit

Controls how many relevant memory notes may be selected in student mode. It is not a limit on every type of record.

Answer size

Estimated output helps price the comparison. The maximum output setting limits a live answer and informs planning.

For this walkthrough

The main demo uses 160 lessons, local planning, a memory limit of 12 and a maximum answer size of 1,200 tokens.

Enlarged view of the actual request controls.

Demo 01 • A yearly-summary request.

The first demonstration uses the application's generated learning history and the same question used by its built-in self-check.

THE REQUEST USED IN THE TEST

"summarize me everything i learned this year, with the dates of the lessons."

160

lessons

720

chat records

42

projects

520

code submissions

280

memory notes

32

assessments

WHAT THE RAW INPUT CONTAINS

Much more than lesson titles.

Repeated structures, learning preferences, scores, conversations, project records and code history all contribute to the full source.

WHAT THE PREVIEW IS TESTING

How much context gets sent.

The application builds a smaller, question-specific input. It can measure the difference without generating an answer.

Synthetic, not student evidence These are generated demonstration records. The test is about input preparation, not a verified report about a real learner or a real school year.

The selection has a visible reason.

The planner adapts to the question. This yearly-summary request selects five categories; a debugging request instead brings in code and mistake history.

Automatic memory slice planner

Copy

```
"exactLessonList": false,
"compactReportRequired": true
},
"selectedSlices": [
  "skill_progress",
  "preference_index",
  "lesson_timeline_monthly",
  "project_ledger",
  "assessment_summary"
],
"reasons": [
  "The request asks for learning history/summary and may require dates, so include a lesson
ledger instead of only memories."
]
},
"activePlan": {
  "planner": "local_rules",
  "intent": "history_report",
  "confidence": 0.9,
  "detail": "compact_monthly_report",
```

Planner tab, scrolled to the selected sections and the recorded reason.

WHAT THE FIVE SECTIONS MEAN

Skill progress

A compact picture of strengths and developing skills.

Learning preferences

Context about how guidance should be presented.

A monthly lesson timeline

Grouped history rather than every full lesson record.

Projects and assessments

Work completed and checkpoints that help explain progress.

Dates need a coverage check This run chose a monthly timeline to fit the answer budget. A summary with dates is not automatically a complete list of every lesson date.

An identity card. A map of the history.

Two different views explain how the source is organized before the selected pieces become a final prompt.

BASE / SOURCE CAPSULE

Base / source capsule

Copy

```
This tiny student capsule is always sent with smart requests:

STU{n:Rony,a:15,lang:hebrew,m:12,lvl:11,g:4,xp:8420,L:160,P:35/42}
SK{var:0.78,in:0.78,if:0.72,bool:0.69,for:0.75,while:0.74,nest:0.76,list:0.72,str:0.79,func:0.79
,param:0.73,dbg:0.72,decomp:0.77,file:0.77,dict:0.78,oop:0.58}
STYLE{tone:friendly_hebrew,pace:step_by_step,ex:games/apps/football/visual
metaphors,avoid:full_solution_first/advanced_concepts_without_bridge/large_code_dumps}

Skill vector:
{
  "var": 0.78,
  "in": 0.78,
  "if": 0.72,
  "bool": 0.69,
  "for": 0.75,
  "while": 0.74,
  "nest": 0.76,
  "list": 0.72,
  "str": 0.79,
  "func": 0.79,
  "param": 0.73,
  "dbg": 0.72,
  "decomp": 0.77,
  "file": 0.77,
  "dict": 0.78,
  "oop": 0.58
}
```

The generated student's compact base capsule and skill values.

Small standing context.

The capsule carries compact background information. For an uploaded file, it instead identifies and summarizes that source.

COMPRESSED INDEXES

Compressed indexes catalog + selected slices

Copy

```
{
  "compressedIndexCatalog": [
    {
      "name": "skill_progress",
      "description": "Aggregated mastery per Python skill, first/last seen dates,
strong/usable/shaky/weak bands.",
      "tokens": 361,
      "chars": 632,
      "preview":
"skills{strong:;usable:var/in/if/for/while/nest/list/str/func/param/dbg/decomp/file/dict;shaky:b
ool/oop;weak:}\nvar:0.78@2025-09-04..2026-12-07;in:0.78@2025-09-04..2026-12-07;if:0.72@2025-09-
07..2026-12-10;bool:0.69@2025-0"
    },
    {
      "name": "lesson_ledger_full"
    }
  ]
}
```

An index catalog entry shows its purpose, estimated size and a preview.

A catalog, not the whole request.

Indexes organize topics, timelines and other records. The catalog shows what is available; selected sections show what will actually be included.

Shorter labels, same intended meaning Abbreviations and compact tables reduce repeated formatting. Their meaning must remain interpretable; this is not ordinary ZIP-style file compression.

Follow the evidence into the request.

The retrieval trace records what was gathered, why it was gathered and the estimated size of each selected section.

Optional AI/tool retrieval trace

Copy

```
"The request asks for learning history/summary and may require dates, so include a lesson ledger instead of only memories."
]
},
"toolTrace": [
  {
    "tool": "get_skill_progress",
    "slice": "skill_progress",
    "reason": "Skill progress is always useful.",
    "chars": 632,
    "tokens": 361
  },
  {
    "tool": "get_preference_index",
    "slice": "preference_index",
    "reason": "Teaching preferences are always useful.",
    "chars": 335,
    "tokens": 66
  },
  {
    "tool": "get_lesson_timeline_monthly",
```

Actual retrieval trace from the generated-history preview.

SELECTED INFORMATION	TOKENS
Skill progress	361
Learning preferences	66
Monthly lesson timeline	2,394
Project history	1,692
Assessment summary	914

Section estimates only. The complete prepared request also includes the capsule, question and other formatting.

Inspectable preparation.

This trace comes from the application collecting sections on the server before an answer call. It is not evidence that an AI searched the web or a separate database.

Two inputs. The same question.

The final comparison is visible: the original full source on one side, and the smaller prepared context on the other.

NORMAL / RAW PROMPT

Original source

SMART / COMPILED PROMPT

Prepared context

Normal raw prompt

Copy

USER:
RAW_STUDENT_JSON:
{
 "generatedAt": "2026-09-09T18:41:58.120Z",
 "datasetSize": "huge",
 "profile": {
 "studentId": "s-rony-001",
 "name": "Rony",
 "age": 15,
 "language": "hebrew",
 "activeMonths": 12,
 "country": "Israel",
 "petName": "Gutchi",
 "gutchiStage": 4,
 "level": 11,
 "xp": 8420,
 "coins": 3240
 },
 "teachingStyle": {

Smart compiled prompt

Copy

USER:
COMPRESSED_STUDENT_CONTEXT_V3:
META {"v":"3.0","domain":"aigutchi_student_memory","stats":
{"lessons":160,"projects":42,"conversations":720,"codeSubmissions":520,"memories":280,"assessmen
ts":32},"intent":"history_report","detail":"compact_monthly_report","range":"full_year","selecte
d":
["skill_progress","preference_index","lesson_timeline_monthly","project_ledger","assessment_summ
ary"],"decode":"STU=profile, SK=skill mastery, SLICE=index selected from full student memory.
Answer directly from STU/SLICES only."}
STUDENT
STU{n:Rony,a:15,lang:hebrew,m:12,lvl:11,g:4,xp:8420,L:160,P:35/42}
SK{var:0.78,in:0.78,if:0.72,bool:0.69,for:0.75,while:0.74,nest:0.76,list:0.72,str:0.79,func:0.79
,param:0.73,dbg:0.72,decomp:0.77,file:0.77,dict:0.78,oop:0.58}
STYLE{tone:friendly_hebrew,pace:step_by_step,ex:games/apps/football/visual
metaphors,avoid:full_solution_first/advanced_concepts_without_bridge/large_code_dumps}

SLICE skill_progress
skills{strong;;usable:var/in/if/for/while/nest/list/str/func/param/dbg/decomp/file/dict;shaky:bo
ol/oon:weak:}

Formatting is reduced

Repeated labels and structures are represented more compactly.

Content is selected

Relevant sections replace a default resend of all history.

Nothing is hidden from review

Both prepared inputs can be inspected before an AI call.

These are cropped screen previews, not the complete inputs. Their displayed heights do not represent their total token counts.

One synthetic test. A 97.7% reduction.

The main dashboard preview was reproduced locally, and the application's built-in self-check also passed.

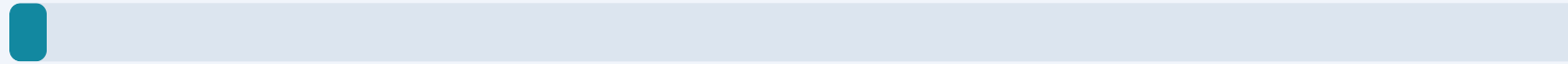
Original full-history input

250,611 tokens



Prepared input

5,855 tokens



Bars show estimated input size on the same linear scale. Figures above match the supplied dashboard screenshot.

42.8x

smaller estimated input

244,756 tokens removed
in this particular preview.

COMPILED TOTAL INPUT

5,855

\$0.002342 input / \$0.003782 est. total

RAW FULL-HISTORY INPUT

250,611

\$0.100244 input / \$0.101684 est. total

SAVED PER REQUEST

244,756

97.7% less input · \$0.097902 input saved

COMPRESSION RATIO

42.8x

Compressed by 97.7%
demo · 160 lessons · 720 chats · 42 projects ·
520 code · 280 memories

Actual dashboard metrics, including the same 97.7% rounded reduction.

Why the self-check differs by five tokens

The self-check reports **250,616 → 5,860**. It counts an empty system-message wrapper that the dashboard omits: five extra tokens on both sides. Both remove 244,756 tokens and round to 97.7%.

What this does not prove It does not establish equivalent answer quality, 97.7% lower total cost, or the same reduction on every dataset.

Demo 02 · Bring your own structured data.

The project extends beyond generated student histories. It can prepare pasted or uploaded JSON and show how it interpreted the source.

Custom examples

School LMS sti

Upload JSON file

Choose File N...

Load example

Format JSON

Clear

Loaded small built-in example into the textarea.

Raw custom JSON source

```
{
  "student": {
    "id": "s-rony-001",
    "name": "Rony",
    "age": 15,
    "language": "he"
  },
  "schoolYear": "2025-2026",
  "lessons": [
    {
      "lessonId": "lesson-01",
```

Large uploaded JSON files are sent as raw request bodies and preview panels are truncated on purpose. Metrics and API calls still use the full source.

Actual custom-source controls with the built-in school example.

What is JSON?

It is a text format software uses for structured information: named fields, lists of records and nested details. Think of it as a machine-readable collection of organized facts.

- 1 Paste a source
- Put valid JSON directly into the text box and format it for easier inspection.
- 2 Upload a file
- Choose a .json file, or a .txt file containing valid JSON. Larger uploads do not need to be rendered in full in the browser.
- 3 Try a built-in example
- The interface includes a school learning-system structure, a chat export and a mixed customer/learning-system example.

Not just student histories

Source-aware handling also covers workshop lessons, tasks, rubrics and instructor notes. The raw file remains the comparison baseline.

See how the file was understood.

The Smart Compiler view exposes a mapping report, including where a record came from and which fields were recognized.

Generic JSON → compiled smart source

Copy

```
],
"mappingsPreview": [
  {
    "sourcePath": "$.lessons[0]",
    "kind": "lesson",
    "date": "2025-09-05",
    "title": "variables + input/output",
    "topics": [
      "py.variables",
      "py.input"
    ],
    "score": 62,
    "summary": "Rony practiced variables + input/output. He needed examples before abstract explanation and one small checkpoint at a time."
  },
  {
    "sourcePath": "$.lessons[1]",
    "kind": "lesson",
```

Built-in school example: the mapping report is scrolled to its first lesson records.

Programmed matching, not guaranteed understanding

Unfamiliar structures may be interpreted incorrectly. Check the mapping report; known production sources benefit from explicit, source-specific adapters.

READ THIS RECORD IN PLAIN LANGUAGE

Original location

The first item in the source’s lessons list. A source path makes the interpretation traceable.

Recognized as a lesson

The report identifies the record type rather than treating every field as unrelated text.

Extracted details

5 September 2025; “variables + input/output”; score 62; related topics and a summary.

Do the maths before asking the model.

For supported numerical questions, the application computes figures from processed records, then includes those figures in the prepared context.

ADDITIONAL SYNTHETIC EXAMPLE · 240 ACTIVITY RECORDS

“How many activity records are in each track, and what is their total duration in minutes?”

Smart compiled prompt

Copy

```
SLICE aggregate_index
AGG_TABLE A1 path=$.activities rows=240;objects=240;processed=240
DISCOVERED_COLUMNS sampled=240; categorical=trck,rcrdd; pairable=trck; numeric=drtnm,scr
FIELD_SAMPLES drtnm:num:u3:"30" | rcrdd:text:u240:"ACT-001" | scr:num:u40:"60" |
stts:text:u1:"completed" | trck:text:u3:"Python"
NUMERIC_TOTALS
- drtnm count=240;sum=10800;avg=45;min=30;max=60
- scr count=240;sum=19080;avg=79.5;min=60;max=99
TOP_VALUE_COUNTS
by track: "ACT"-80: "Python"-80: "Web"-80:
```

Actual aggregate section in the prepared prompt. “processed=240” confirms the rows covered here.

The short field labels are explained by the compiler’s dictionary: **trck** means track; **drtnm** means duration in minutes.

No AI answer was generated The results at right are a readable transcription of the application’s computed counts and grouped totals.

TRACK	RECORDS	MINUTES
Python	80	2,400
AI	80	3,600
Web	80	4,800
Total	240	10,800

An AI can then explain the supplied figures instead of adding up a pile of raw rows itself.

This example processed all 240 rows. Other sources may encounter configured processing limits, which must be checked.

Useful before an AI is connected.

The local preview is a working part of the product, not a fabricated answer. Live AI planning and answer generation are optional.

WITHOUT AN AI API KEY

Prepare, calculate and compare.

Organize the source, build compact indexes, calculate supported figures, choose sections with local rules and inspect token/cost estimates.

WITH A CONNECTED AI SERVICE

Add planning or an answer.

Run an optional AI planner, or send the prepared or raw input to the answer model. A failed planner API request falls back to local planning.

API response comparison

Copy all

Request cost summary

LTR technical metadata

MODE: smart
OPENAI_API_KEY missing — local mock only.
ESTIMATED FINAL INPUT TOKENS: 5,855
ESTIMATED FINAL INPUT COST: \$0.002342

AI output

Auto RTL/LTR, code stays LTR

OPENAI_API_KEY is not set. This is a local mock response. Add your key in .env and restart the server to call OpenAI.

No invented model output.

This PDF does not present the no-key message as an AI answer or as evidence of answer quality.

One response view, two send options.

Each send updates the same response panel. Copy a result before running the other version for comparison.

Readable multilingual answers.

The response display handles English and Hebrew direction while keeping code left-to-right.

Genuine no-key response state: the interface explicitly labels it "local mock only".

Input savings are not the whole bill.

The dashboard separates input estimates from estimated total cost. Optional planning also consumes tokens and must be counted.

ILLUSTRATIVE COSTS FROM THE SUPPLIED APP

PREVIEW ESTIMATE	RAW INPUT	PREPARED INPUT
Input portion	\$0.100244	\$0.002342
Output assumption	\$0.001440	\$0.001440
Estimated total	\$0.101684	\$0.003782

This preview assumes 900 output tokens and no paid planning call. It uses the app’s configured rates: \$0.40 per million input tokens and \$1.60 per million output tokens.

Not a current pricing quote These are snapshot estimates from the supplied application, not verified current provider prices or an invoice.

- 1

Count the planning step
When a successful AI planning call is used, the comparison includes its input, output and configured-rate cost—not only the smaller final request.
- 2

Distinguish estimates from usage
Before a call, token counts are estimates. After a live call, the application can show usage reported by the API.
- 3

Keep monetary figures in context
Even “actual cost” displays are calculated from token usage and configured rates. They are not a fetched billing statement.
- 4

Match the settings to the comparison
Model choice, rate overrides and the assumed answer length affect the totals. Review them before using the result for budgeting.

Large inputs need clear boundaries.

The application includes controls for large uploads and responsive previews. A smaller on-screen preview must not be confused with complete processing.

01 · TRANSFER

Upload boundary

Large JSON can be sent without filling the browser's text box. Server-side request-size limits still apply.

02 · PREPARATION

Processing boundary

The compiler caps records, sampled table rows, aggregates and other structures. Calculations cover processed rows, not an unlimited dataset.

03 · PREVIEW

Display boundary

Long prompt and section previews may be shortened for the browser. The raw comparison is designed to measure the complete uploaded source.

WHERE INFORMATION GOES

Local does not always mean “my device”.

Preparation runs on the application server. In a local setup that server is on the same machine. A live AI request sends the selected prepared context—or the raw source, for the raw option—to the AI service.

OPERATIONAL CONTROLS

Limits and failures are part of the product.

The supplied code includes request limits, timeouts and error handling. Very large sources can still exceed server resources or a selected model's limits. These controls are not a security or compliance certification.

For large sources, check processed-row counts, truncation markers, mapping quality and preview warnings before relying on a result. Very long inputs may use sampled token estimates.

An application, not just a prompt.

The project includes the interface, processing logic, information-selection system, AI integration and the comparison dashboard.

COMPLETE PROJECT OWNERSHIP

"I built the complete application: the screens people use, the data-processing logic, the information-selection system, the AI connection, and the comparison dashboard."

The distinctive idea is to combine **smaller formatting** with **question-specific selection**—and make both visible.

The supplied design does not require a database, a separate AI search database or a large frontend framework.

01 The user experience

Upload screens, source controls, inspection tabs and readable responses.

02 The preparation engine

Data interpretation, compact structures, supported statistics and summaries.

03 The selection system

Local rules, optional AI planning and a trace of retrieved sections.

04 The AI connection

Prepared and raw answer requests, usage display and failure handling.

05 The comparison dashboard

Input size, savings, cost estimates and planning overhead in one view.

The technology, in plain language.

Each component has a practical job. The main data preparation and calculation logic was implemented directly in JavaScript.

TECHNOLOGY	ITS ROLE IN THE PRODUCT
JavaScript + Node.js	Implemented the custom processing and selection logic, and ran it on the application server.
Express	Connected the browser to uploads, previews and answer requests.
HTML, CSS + browser features	Built the screens, controls and response display, including text-direction handling.
JSON	Provided the structured format used for source data and prepared information.
OpenAI Responses API	Connected the optional AI planning step and final answer generation.
js-tiktoken	Counted or estimated the text units used by the original and prepared inputs.
Helmet, CORS + express-rate-limit	Added web-response settings, cross-origin request configuration and request-rate limits.
compression	Reduced web-response transfer size. This is separate from AI-input compression.
dotenv, npm + Git	Supported configuration, software dependencies and version tracking.

What the demo proves—and what it does not.

The application demonstrates inspectable preparation and measurable input reduction. Other outcomes need their own validation.

DEMONSTRATED

Input size was reduced

The synthetic 160-lesson self-check passed and produced a 97.7% reduction in estimated input tokens.

DEMONSTRATED

Selection is inspectable

The source mapping, plan, retrieval trace and prepared prompt can be opened and reviewed.

DEMONSTRATED

Supported figures are computed

The additional 240-row example produced the expected counts and duration totals before any AI answer.

REQUIRES VALIDATION

Answer quality is not established

No live AI answer comparison or quality evaluation was performed for this document. More compression is not automatically better.

REQUIRES REVIEW

Coverage is not guaranteed

Mappings can be wrong; selected sections can omit details; processed data can be capped. Check the evidence and the required detail.

DEPENDS ON THE CASE

Savings are not universal

Small sources can become larger because prepared structures add overhead. Model settings and the question also change the outcome.

A useful counterexample In the built-in school example used on pages 13–14, 3,662 raw tokens became 9,899 prepared tokens. The per-request comparison matters; savings should never be assumed.

An eight-minute demo, ready to present.

Use the pages below as your visual script. No online access or account credentials are needed to explain the product.

TIME	PAGES	SHOW	SAY
1 min	04	Show the workspace	"This screen lets us inspect the entire preparation process, not just a final number."
1 min	07	Introduce the large history	"We will ask one summary question over a generated dataset containing lessons, chats, projects and more."
1 min	08–10	Explain the selection	"The system chooses categories of evidence, and we can inspect what it collected and why."
1 min	11–12	Compare the two inputs	"The synthetic test reduced estimated input tokens by 97.7%; that is an input-size result, not a quality guarantee."
2 min	13–15	Show another type of data	"The same approach can prepare uploaded JSON and calculate supported figures before asking an AI to explain them."
2 min	16–17, 21	Close with the boundaries	"The local preview works without an AI key. Live answers, total cost and data coverage are separate things to evaluate."

A DESCRIPTION YOU CAN REUSE

I built AI Token Compressor to organize large datasets, calculate supported statistics and select relevant evidence before an AI is asked to answer. It makes that preparation visible and compares input size and estimated costs, including optional planning. In one synthetic local test, it reduced estimated input tokens by 97.7%.

The terms, translated.

The interface uses technical labels. These are the ideas behind them.

Token	A small unit of text an AI processes. It is not necessarily a whole word.
Context	The information supplied alongside a question so the model can answer it.
JSON	A text format for structured data: named fields, records and lists.
Compiled input	The prepared version of the source that is assembled for a particular question.
Capsule	A small profile or source summary included as basic context.
Index	An organized, compact representation that helps locate or summarize records.
Slice	One selected section of information included in the prepared request.
Planner	The logic that chooses the information categories needed for a question.
API	A connection that lets the application request a service from another system.

FOUR DISTINCTIONS TO KEEP IN MIND

Input ≠ output

Input is what the model receives. Output is the answer it generates.

Preview ≠ full source

A shortened browser preview does not establish how much data was processed.

Reduction ≠ answer quality

A smaller input still needs enough correct evidence for the question.

Reported usage ≠ an invoice

API-reported tokens can inform a cost estimate; provider billing is a separate record.

How this document was grounded.

Based on the supplied application archive and screenshot. Local checks and captures: 9 September 2026.

S1	Supplied project brief & original screenshot AI Token Compressor description and the dashboard image supplied in this conversation.
S2	Project documentation README.md
S3	The browser interface public/index.html · public/app.js · public/styles.css
S4	Question planning & selected sections src/services/contextArchitecture.js
S5	Source interpretation & calculations src/services/universalDataCompiler.js
S6	Prompt assembly & comparison inputs src/services/payloadBuilder.js · promptMessages.js · externalJsonPromptBuilder.js
S7	Token counts & cost calculations src/services/tokenizer.js · promptMetrics.js · costs.js
S8	Application server, AI connection & dependencies src/server.js · src/services/openaiClient.js · package.json
S9	Built-in self-check, rerun for this document src/utils/selfCheck.js · npm run check
S10	Additional local walkthrough captures Generated-history preview, built-in school example, no-key state and a new 240-row synthetic activity dataset.

Archive: aigutchi-context-router.zip. Unqualified service filenames are in src/services/. Page footers reference the source groups above.

LOCAL CHECK · PASSED

160 lessons · 250,616 → 5,860

The included self-check completed successfully. It verifies a lesson-history selection and checks that a football-app request does not falsely trigger all-history or for-loop context.

What the screenshots represent

The original supplied screen and new captures of the supplied application’s local workflow. Some panels are cropped or scrolled to make relevant details readable. Data shown is synthetic.

What was not tested here

No live AI answers, comparative answer-quality study, production security assessment or current provider-price verification. No placeholder response is presented as a real AI result.

RETURN TO CONTENTS ↗