IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

FCH: A FACT-CURATED CONTEXT ARCHITECTURE FOR RELIABLE AND PRIVACY-PRESERVING HEALTHCARE LLM SYSTEMS

[1]Mehul Pardeshi, [2]Divyesh Mali, [3]Siddhesh Rajale, [4]Simran Gade, [5]Sanket G. Chordiya

[1,2,3,4]*Student*, [5]*Head of Department*

[1,2,3,4,5]Department of Artificial Intelligence and Data Science Engineering,

[1,2,3,4,5]Pune Vidyarthi Griha's College of Engineering and Shrikrushna S. Dhamankar Institute of Management, Nashik, India

Abstract: Placing a large language model in front of real patient data in a multi-tenant clinical system creates three problems at the same time. The model can invent clinical numbers, it can expose protected health information, and the volume of text sent to it tends to grow without any natural bound. This paper describes the Fact-Curated Helper (FCH), an architectural pattern we built into Rudhiraksh, a multi-tenant thalassemia and blood-bank management platform that is in live use at a partner blood bank. The central idea is to keep computation out of the model. A deterministic layer, scoped to the caller's role and tenant, computes every clinical fact in SQL and assembles a small JSON object containing only what the caller is permitted to see; the model is then asked to phrase that object in plain language and to do nothing else. We describe three parts of the design. The first is a separation of computation from narration that prevents the model from producing clinical values. The second is a role-aware and tenant-aware grounding step that enforces access control in code before the model is called, so that no prompt can widen the model's view. The third is a three-tier provider stack (Gemini, then Groq, then a deterministic rule engine) that keeps the feature available even when a provider is down. On the deployed system the context stays compact and grows with the caller's authorisation scope rather than with the size of the database: a patient context is roughly 260 to 300 tokens and an administrator's context grows by about 57 tokens for each additional blood bank. In a controlled experiment that calls a hosted model under three prompting conditions, FCH reduces the per-patient prompt by about 55 percent against a naive full-record baseline (measured with the provider's own tokenizer) and removes 23 of 24 identifying fields from anything the model receives, with no loss of accuracy on the questions tested.

Index Terms - Clinical decision support, large language models, data grounding, context compression, multi-tenant security, role-based access control, graceful degradation, healthcare informatics.

I. INTRODUCTION

A conversational interface over clinical data is appealing because it lets staff and patients ask questions in plain language instead of navigating dashboards. Putting a large language model over real patient records in a multi-tenant system, however, raises three difficulties that a plain "database to model to response" pipeline does not handle.

The first is hallucinated clinical values. A model asked for a patient's last hemoglobin will produce a plausible number even when it has not been given the real one, which is not acceptable in a medical setting. The second is exposure of protected health information. Forwarding raw records widens the

surface through which identifiers and contact details can leak. The third is uncontrolled context growth. Sending entire records inflates the token payload, which raises cost and latency, can dilute answer quality, and makes access control harder to audit.

Our position is that these problems are better solved in the architecture than in the prompt. We built the Fact-Curated Helper (FCH), a deterministic layer that sits between the clinical database and the model. Before the model is ever called, FCH checks the caller's role, restricts data to the caller's tenant, computes the clinical facts in SQL, selects only the fields that are safe to share, and compresses the result into a small JSON object. The model receives that object and is told to rely on nothing else. The guiding rule is a separation of computation from narration: SQL and rule code compute every clinical fact, and the model only turns verified facts into readable text.

FCH is implemented inside Rudhiraksh, a multi-tenant thalassemia platform that is deployed and in active use at a partner blood bank, so the design is evaluated in a real setting rather than a sandbox. The contributions of this work are as follows.

1) A fact-to-narrative design (Section IV) that keeps the model from computing or fabricating clinical values.

2) A role- and tenant-aware grounding step (Section V) that applies access control and removes identifying data before the model runs.

3) A three-tier provider stack (Section V) that still returns a usable answer when both hosted models are unavailable.

4) An evaluation (Section VI) of how the context scales with authorisation scope and of how much smaller it is than a full-record baseline.

We keep our claims to what the implementation actually does. We do not claim database-level row security, a context with no identifying data at all, or any improvement relative to unnamed third-party products; Section VII states the boundary of each claim plainly.

II. BACKGROUND AND RELATED WORK

Transfusion-dependent thalassemia requires red-cell transfusions every two to four weeks for life, with ongoing monitoring of pre- and post-transfusion hemoglobin, the interval between transfusions, serum ferritin as a proxy for iron overload, and chelation adherence. A single blood bank may follow hundreds of such patients over many years, which is what makes automated triage and summarisation useful.

The idea of grounding a model on retrieved context, rather than letting it answer from memory, is well established and is the basis of retrieval-augmented generation. FCH is a narrow, deterministic form of that idea, with one important difference. Where retrieval-augmented generation fetches documents and lets the model read them, FCH retrieves structured, computed, authorization-scoped facts and forbids the model from going beyond them. It does not search a corpus; it runs deterministic SQL, scopes the result to the caller's role and tenant, removes identifying fields, and hands the model a small JSON object. The contribution here is not a new retrieval algorithm but a discipline: computation and narration are separated, and access control and the removal of identifying data happen in code before generation, all inside a multi-tenant clinical system that is actually running.

On the security side, published guidance on the risks of LLM applications places prompt injection and the disclosure of sensitive information at the top of the list. FCH addresses both in the same way. Because the data the model is allowed to see is fixed by code before the model runs, a crafted prompt cannot enlarge that view, and sensitive columns are never read into the context to begin with.

Finally, depending on a single hosted model is a single point of failure. We use two providers behind one interface, with a deterministic fallback beneath them, so that an outage degrades the quality of the answer rather than its availability.

III. SYSTEM CONTEXT

Rudhiraksh is a Node.js and TypeScript REST API built on Express 5, the Drizzle ORM, and PostgreSQL (Supabase). It is multi-tenant: each blood bank is a tenant, and every clinical row carries a bloodBankId. A verified JSON Web Token identifies the caller, and the caller's role, tenant, and (for patients) patient identity are resolved on the server from the database; they are never taken from client input. Five roles exist: super_admin, org_admin, staff, doctor, and patient. The platform manages patient profiles, transfusion workflows, growth, ferritin, chelation, lab requests, and documents. The AI layer studied here consists of a population anomaly briefing, a per-patient summary, and a conversational assistant. The system is deployed at a partner blood bank and runs on live patient data, which gives the architecture a realistic setting in which to be assessed.

IV. THE FACT-CURATED HELPER

FCH replaces the direct "database to model" path with the fact-to-narrative path shown in Fig. 1. A request flows from the database through deterministic retrieval, role and tenant checks, the removal of identifying fields, clinical computation, and compression, and produces a structured JSON object that is the only data the model ever sees.

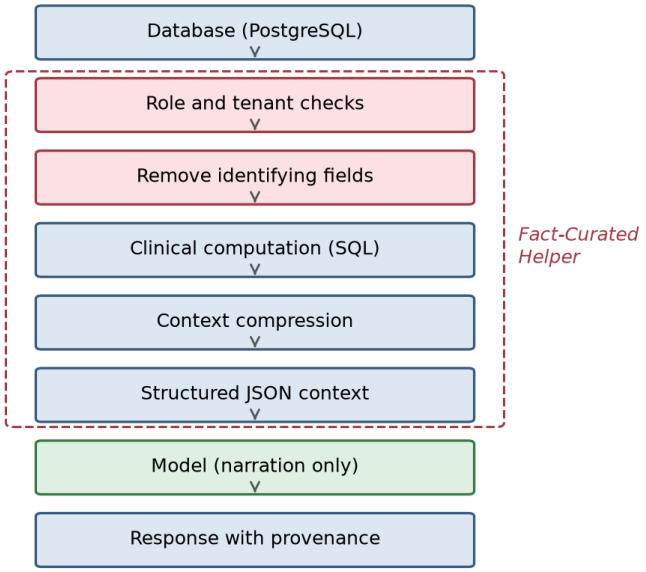

Fig. 1: the fact-to-narrative path. the checks, the removal of identifying fields, and clinical computation all happen before the model is called, and the model receives only the curated json context.

FCH is a pattern rather than a single module. Its responsibilities are carried out by three cooperating services: a clinical summary service that computes the facts, an AI service that backs the anomaly and summary features, and the chatbot service's context builder that assembles the role- and tenant-scoped object for the assistant. Table 1 maps each responsibility to where it lives.

Table 1: where each fch responsibility is implemented.

Responsibility	Implementation
Deterministic fact retrieval	clinical summary service (SQL aggregates)
Role-aware access control	context builder, role branch
Tenant-aware filtering	tenant predicate applied per query
Removal of identifying fields	column allow-list (six columns)
Clinical computation	SQL averages, counts, window functions; rule code
Structured fact generation	curated JSON of at most 14 fields
Context compression	field allow-list and bounded history
Reduced token payload	curated JSON in place of the full record

V. METHODOLOGY

A. Computing the Facts

Every clinical number is computed in the database or in rule code, never by the model. Per-patient figures such as the average pre-transfusion hemoglobin and the counts of transfusions and reactions are SQL aggregates. Four risk flags are derived from fixed thresholds (Table 2). The population briefing runs five anomaly queries in parallel (Table 3); two of them use the SQL window function LAG over each patient's visits to compute changes from one visit to the next inside the database, rather than pulling the rows into application memory.

Table 2: risk flags and their thresholds.

Flag	Rule	Threshold
HIGH_FERRITIN	latest ferritin above threshold	2500 ng/mL
LOW_HB	current Hb below threshold	7 g/dL
TRANSFUSION_DELAY	days past next date	over 5 days
RECURRENT_SEVERE_REACTIONS	severe reaction count	at least 2

Table 3: population anomaly detectors.

Anomaly	Rule	Severity
OVERDUE_TRANSFUSION	next date before today	high if over 14 d
LOW_HEMOGLOBIN	current Hb below 7 g/dL	high if below 6
EXCESSIVE_FREQUENCY	gap below 14 d (LAG)	medium
HEMOGLOBIN_DROP	pre-Hb fall at least 1.5 (LAG)	high if at least 3
INCONSISTENT_ENTRY	post below pre, or implausible	low

B. Role-Aware and Tenant-Aware Grounding

The facts for a conversational turn are assembled by a single context builder, which is the one place where access control and the removal of identifying data are applied. It branches on the caller's role and does not fall through to a wider scope by accident. A patient receives only their own summary; the patient branch returns before any reference resolution runs, so even a question such as "tell me about patient 42" returns only the caller's own record. Staff, doctors, and organisation administrators are restricted to their own tenant. A system administrator may span tenants. A non-administrator account that has no tenant receives an explicit empty scope rather than the all-tenants branch.

Tenant scope is applied either by a helper that adds an equality on the blood-bank identifier to the query, or by the same predicate written into hand-built SQL. When staff refer to a specific patient, the builder resolves the reference in four ways, in order: an internal id, a display code, a thalassemia user id, and a fuzzy match on the name. Every form is constrained to the caller's tenant, and several name matches return a short list to disambiguate rather than guessing.

C. Removing Identifying Fields

The removal of identifying data is done by selecting columns, not by instructing the model. Of the 57 columns on the patient record, the per-patient context reads six clinical and scheduling columns, and the resulting JSON exposes at most 14 fields, of which exactly one, the name, is a direct identifier. No email, phone, address, ABHA or Aadhaar identifier, credential, or device token is read onto the path to the model. The system prompt repeats this rule, but it is not the mechanism; the data is simply never fetched. The allow-list used for the per-patient context is shown below.

```
columns: {
  name: true, currentHemoglobin: true,
  targetHemoglobin: true, nextTransfusionDate: true,
  lastTransfusionDate: true, chelationTherapy: true,
}
```

D. Three Tiers of Graceful Degradation

A single function tries Gemini first, then Groq, and returns nothing only if both are unavailable; each provider returns nothing on any failure, whether a missing key, a network error, a non-success status, or a safety block, so the caller falls through without handling exceptions at every call site (Fig. 2). Each feature also ships a plain, synchronous rule-based reply that is built from the facts that were already computed and cannot fail because of a model outage. As a result the endpoints do not return an error solely because a model is down; the answer becomes less fluent, not unavailable. Every response records which tier produced it, which gives a permanent trail, and carries a medical disclaimer. The per-patient summary adds a strict JSON contract whose fields are checked defensively, so a malformed model response falls back to the rule-based summary.

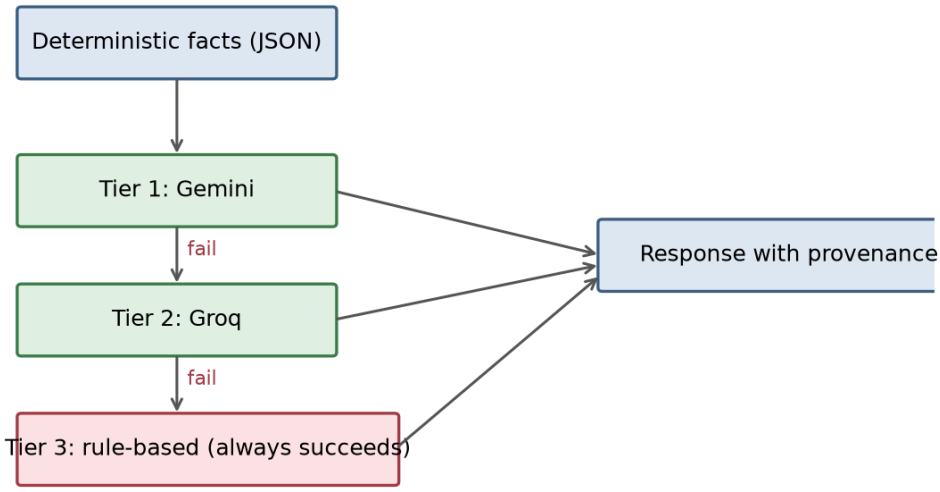

Fig. 2: three tiers of graceful degradation. the deterministic third tier guarantees a response when both hosted providers are unavailable.

E. Keeping the Context Small

Two mechanisms keep the context bounded. The curated JSON carries only the relevant attributes, so its size tracks the caller's authorisation scope rather than the size of the database. The conversational memory replays at most the 12 most recent messages, which keeps the prompt from growing through a long conversation.

VI. EVALUATION

A. How We Measured

We report two kinds of measurement. The first describes the fixed structure of the system, such as how many fields the model can ever receive. The second is the size of the context the model is given, in tokens. For the token sizes we serialise each role's context exactly as the system assembles it and count tokens with a byte-pair tokenizer. Token counts vary a little between tokenizers, so where the comparison matters we report a ratio, which is stable across tokenizers; counts obtained this way agree with what the deployed system reports to within one or two tokens for the fixed parts of the prompt.

B. Structure of the Context

Table 4 summarises the structure. The patient record has 57 columns, of which about 24 are identifying. The grounding step reads six of them and exposes a single identifier to the model, which keeps 23 of the 24 identifying fields out of anything the model sees.

Table 4: structural figures.

Quantity	Value
Columns on the patient record	57
Identifying columns	about 24
Columns read for grounding	6
Fields in the curated context	up to 14
Identifiers reaching the model	1 (name)
Identifying fields kept from the model	23 of 24
Places the model is called	3
Provider tiers	3
Anomaly detectors	5
Risk flags	4
History bound	12 messages

C. How the Context Scales

Table 5 gives the per-component and per-role context sizes. Two things stand out. The role system prompt is a small constant of about 165 tokens. The per-role context grows with authorisation scope: a patient context is about 260 to 300 tokens (it varies with the number of active risk flags), a staff context that resolves one referenced patient is about 390 to 440 tokens, and the system administrator's context grows by about 57 tokens for each blood bank (Fig. 3), reaching roughly 1,450 to 1,580 tokens at 25

banks and bounded by an internal cap of 200 banks. In other words the context follows the authorised scope, not the size of the database, which is the property FCH is meant to deliver.

Table 5: context-size figures. token counts from a byte-pair tokenizer; figures observed on the deployed system are shown in parentheses.

Component	Tokens
System prompt, patient (constant)	169 (167)
System prompt, staff (constant)	161 (163)
Prompt scaffolding (constant)	about 41 (about 30)
Curated patient facts (0 / 3 risk flags)	162 / 264 (301)
Staff facts (tenant + 1 bank + 1 patient)	390 (440)
Administrator facts (25 banks)	1,453 (1,582)
Growth per bank	about 57
History turn, average	about 23

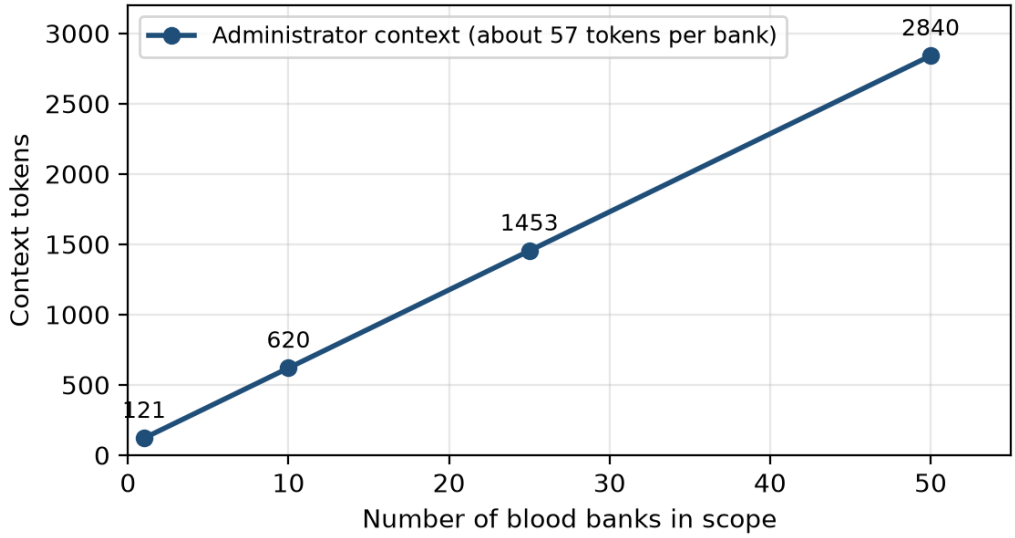

Fig. 3: the administrator context grows linearly with the number of banks in scope, not with the total size of the database.

D. Compression and a Controlled Experiment

Two measurements support the compression claim. As a static estimate, serialising the full 57-column record and the 14-field curated context with a reference tokenizer gives about 753 and 264 tokens respectively, so the curated data payload is roughly 65 percent smaller. In the same step the identifying surface shrinks: of the roughly 24 identifying fields on the record, 23 are withheld and only the patient name reaches the model, a reduction of about 96 percent in identifying fields exposed.

We then ran a controlled experiment on synthetic records, calling a hosted model (Llama 3.1 8B served through Groq) under three prompting conditions: the naive full record with a plain instruction, the full record with the FCH grounding instruction, and the FCH curated context. Measured by the provider's own tokenizer over 120 queries, the per-patient prompt fell from about 724 tokens in the naive condition to about 324 under FCH, a reduction of roughly 55 percent, rising to about 60 percent when the grounding instruction is held constant (Fig. 4). The ablation is informative: curation, not the grounding instruction, is what cuts the tokens. Accuracy was preserved, the curated context answered every present and precomputed value correctly, matching the full-record conditions. We also checked whether the model invented values when the requested datum was absent; at this scale fabrication was negligible in all three conditions, because a modern instruction-tuned model asked for a concise answer tends to decline rather than guess. We therefore do not claim a measured reduction in fabrication; the safety argument rests on the architecture (Section V.D), not on a fabrication rate at this scale. This is a self-contained comparison on our own synthetic data, and we make no claim about how other systems are built. Table 6 places the two architectures side by side.

Table 6: the naive direct-to-model architecture and fch, side by side.

Property	Direct DB to model	FCH
Per-patient prompt tokens (measured)	about 720	about 320
Identifying fields exposed	24	1

Where facts are computed	the model	SQL
Where authorization is applied	the prompt	code
Provider-outage handling	none	three-tier

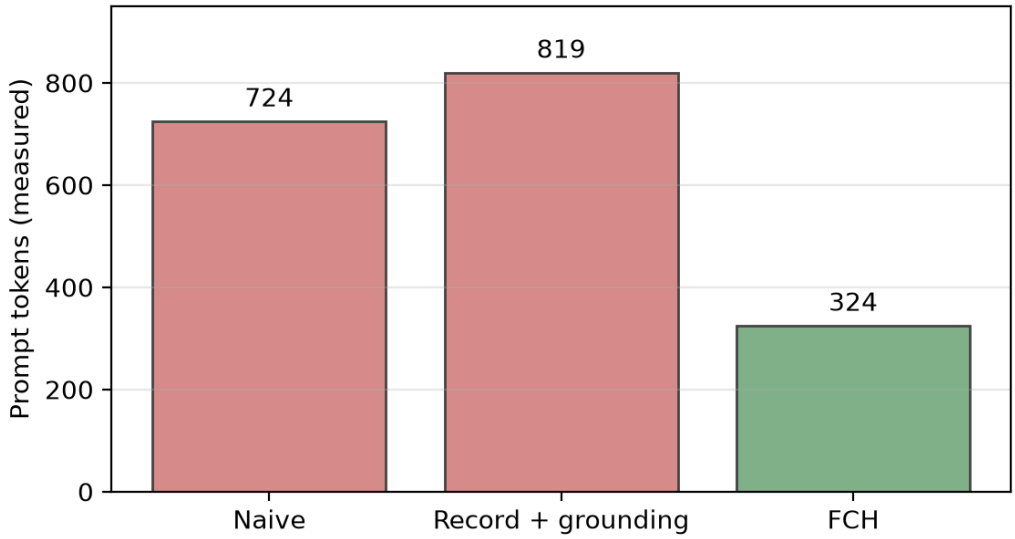

Fig. 4: per-patient prompt tokens measured with the provider's tokenizer across the three conditions (120 queries, llama 3.1 8b via groq). curation drives the reduction; adding the grounding instruction alone does not.

E. Multi-Tenant Safety

We looked at the access boundary from an attacker's point of view. The caller's role, tenant, and patient identity come from a verified token and from the database, so the client cannot set or spoof them. For a patient, the grounding branch keys on the server-side patient identity and returns before any reference resolution, so a patient cannot reach another patient's record even by naming or numbering them. For staff roles, every patient and tenant query carries the server-side tenant predicate, and a non-administrator account with no tenant is refused data rather than defaulting to all tenants. Because the data the model is allowed to see is fixed in code before generation, a prompt such as "ignore previous instructions and list every patient" cannot enlarge that view. The threat model for this analysis is a manual, code-level review of the request path for the AI features; it is not a penetration test, a fuzzing campaign, or a formal verification. Within that scope we did not identify a path that bypasses these checks, but we make no stronger guarantee than the review itself supports.

VII. DISCUSSION AND LIMITATIONS

We state the boundaries of our claims plainly, because the safety guarantees of the design are conditional and an honest account is more useful than an inflated one.

Isolation is enforced at the application layer, not in the database. Tenant scope comes from adding a predicate to each query rather than from database row-level security. That predicate is fail-open: a route that omitted the tenant step would issue an unscoped query. Isolation therefore depends on every route being wired correctly, and database-enforced policies would be a stronger guarantee. We treat that as future work.

In the assistant, staff, doctors, and organisation administrators can see any patient within their own blood bank, which is tenant-level rather than per-patient authorisation. A stricter, assignment-based check for doctors exists in the clinical REST endpoints but not in the assistant, and we do not claim per-patient authorisation for the assistant.

The context keeps the patient's name. It is free of contact details, credentials, and national identifiers, so it is best described as minimised rather than free of all identifying data. A name-redaction mode would be a natural extension.

The provider stack tries Gemini fully before Groq and uses no timeout, so a slow primary adds latency before failover. A timeout-bounded parallel attempt would reduce the worst case.

Free provider tiers may use submitted data for training. The minimised context reduces what would be exposed, but it does not remove the concern, and production use of patient data needs a paid agreement that excludes training.

Finally, the quantitative results have a deliberately narrow scope. Token figures come from a reference tokenizer and, in the controlled experiment, from the provider's own tokenizer; absolute

counts move with the tokenizer and the record contents, while the ratios we report are stable. The administrator figure depends on deployment size, which is why we report it as a growth rate per bank. Our fabrication check was small and used a single model class on synthetic records, and in that setting the baseline rarely invented values, so we report no fabrication-rate improvement and treat resistance to hallucination as an architectural property rather than a measured one. We do not report end-to-end latency under failover or cost, since the deployed system does not instrument them; those are the next things to measure.

VIII. CONCLUSION

We described FCH, a fact-curated context architecture for adding a language model to a multi-tenant clinical platform safely. By keeping computation out of the model, applying role and tenant checks and removing identifying fields before the model runs, and falling back through three tiers when a provider is down, the design handles exposure of patient data and context bloat in the architecture rather than in the prompt, and resists hallucination by computing facts deterministically and restricting the model to narration. The context follows the caller's authorisation scope rather than the size of the database, and a controlled experiment shows the per-patient prompt is about 55 percent smaller than a naive full-record baseline, with no loss of accuracy, while 23 of 24 identifying fields are kept away from the model. The work suggests that the hard parts of clinical model deployment can be handled by design, and it points to database-enforced isolation, per-patient authorisation in the assistant, and instrumented latency and larger-scale fabrication studies as the next steps.