

A Layered Framework for Query-Based Engineering Knowledge Leakage

in AI-Assisted Rocket Design

Yinchien Huang · Daniel DeLaurentis

Purdue University · School of Aeronautics and
Astronautics

PURDUE UNIVERSITY

Operational context: AI-assisted engineering crosses trust boundaries

Not every AI-assisted workflow is local or enterprise-managed

COMMON AI INTERACTION PATHS

Enterprise-managed AI

Private deployment or approved enterprise API

Third-party cloud / API

The design artifact may stay local while the query crosses a vendor trust boundary.

Personal / unmanaged AI

Personal account, personal API key, outside enterprise visibility and governance.

47% of GenAI users
observed used
personal AI apps

Cloud & Threat Report 2026

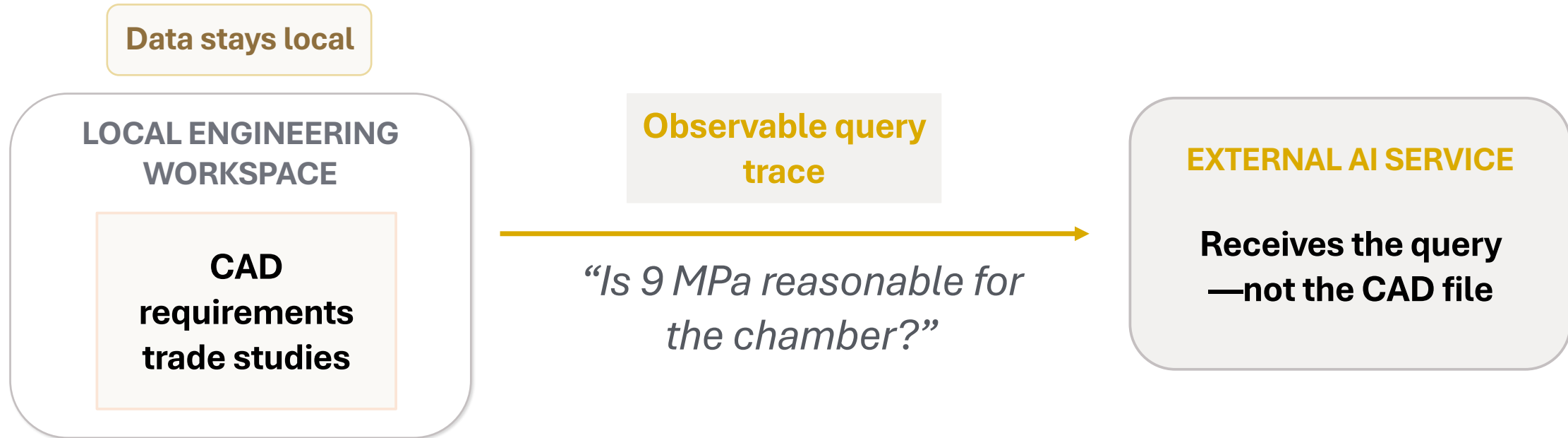
WHY QUERY PRIVACY REMAINS RELEVANT

Even when CAD, requirements, and proprietary files remain internal, ordinary engineering prompts can encode design intent, subsystem relationships, and current design state.

Source: Netskope Threat Labs, Cloud and Threat Report 2026 (usage statistics: Oct. 2024–Oct. 2025).

The privacy gap: the query itself can reveal the design

Sensitive engineering knowledge may leave the local environment through ordinary interaction traces.



From a single query or a sequence, an observer may infer:

**Engineering
intent**

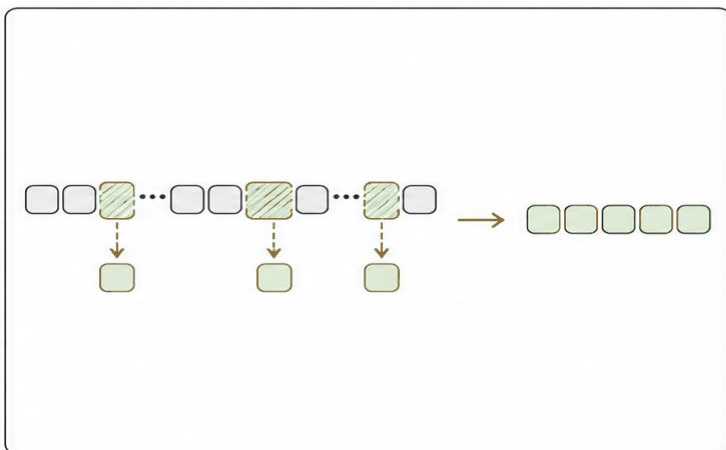
**System
architecture**

**Design
parameters**

**Key shift: privacy risk is not limited to uploaded files
interaction traces can act as a semantic side channel.**

Related work: three major approaches to prompt privacy

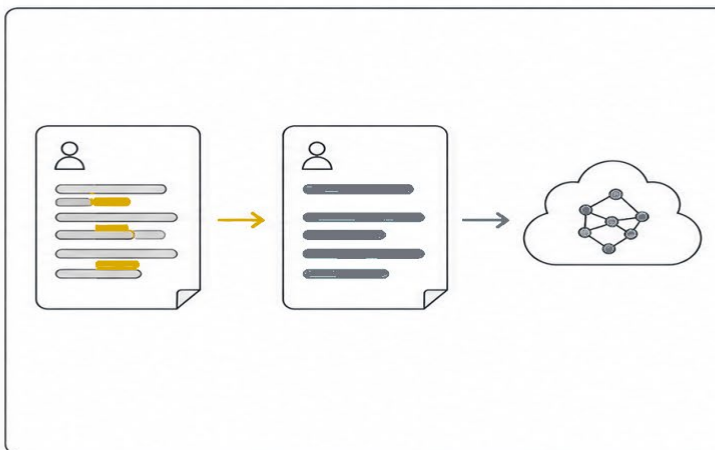
Token-level text sanitization



CusText — Chen et al., ACL Findings 2023

Protects sensitive lexical content while managing privacy-utility trade-offs.

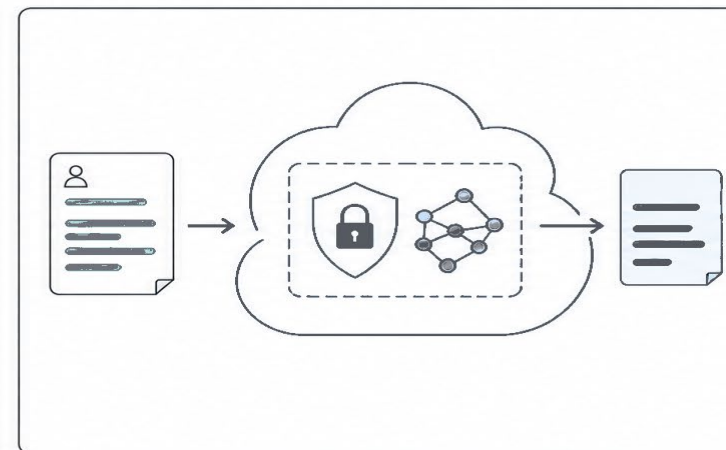
Prompt anonymization / obfuscation



EmojiPrompt — Lin et al., NAACL 2025

Generative / context-aware prompt rewriting before cloud submission.

Confidential inference



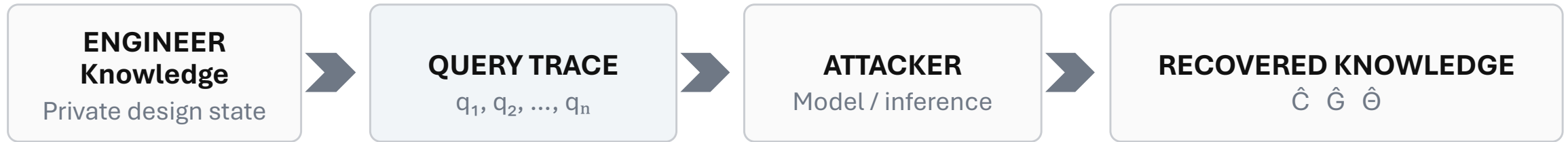
Confidential Prompting — Li, Gim & Zhong

Trusted execution / secure partitioning keeps the prompt hidden from the provider.

This work → layer-specific + sequence-level engineering knowledge recovery

Threat Model: What Can an Observer Recover from Engineering Queries?

The attacker sees the query trace and attempts to reconstruct protected engineering knowledge.



Three leakage layers

CONTEXTUAL

Context / intent

What engineering problem is being solved?

Domain

Intent

Recover the domain, task, and fine-grained engineering intent.

STRUCTURAL

Entity + relation

Which components and functional links are present?

Entity

Relation

Recover system elements and how they interact.

PARAMETRIC

Design values

Which values or constraints define the design?

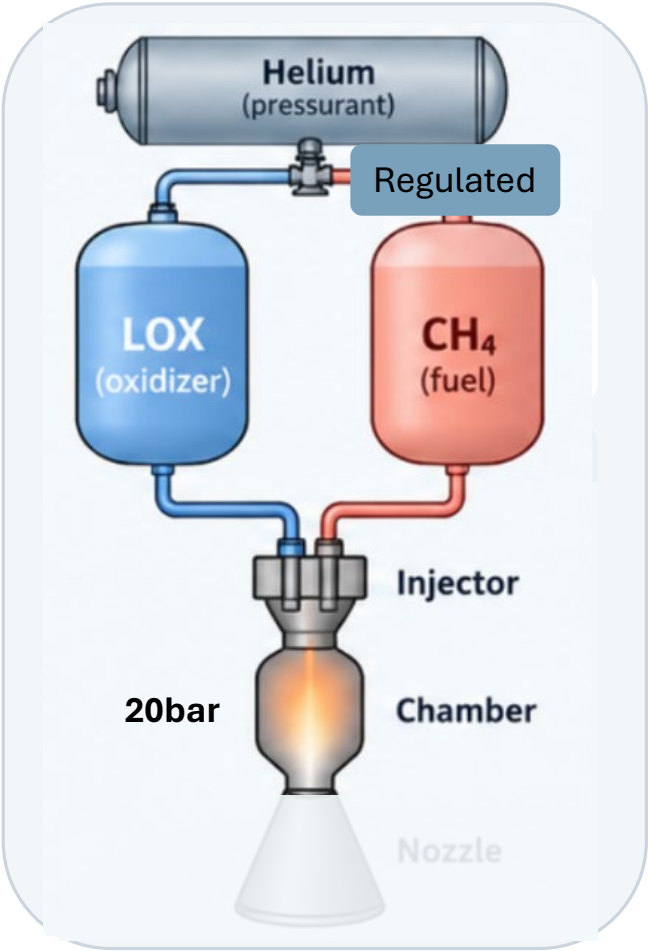
Value

Constraint

Recover numerical design values, operating points, and constraints.

Representative Example: From Query Trace to Design Inference

No single query reveals the full design — the sequence progressively narrows the design space.



Observed query sequence

- 1 What insulation approach would you recommend for LOX and methane tanks on an upper stage?
- 2 For an upper stage this size, when does a pressure-fed system make more sense than using a pump?
- 3 How do I size the helium pressurant if I want to maintain constant propellant feed pressure during the burn?
- 4 If the inlet pressure is about 23 bar and I need roughly a 3-bar pressure drop across the injector, is that enough margin?
- 5 What regulator set pressure should I use to keep the injector inlet pressure above 23 bar throughout the burn?

What an attacker can infer

CONTEXT

LOX/CH₄ + upper-stage application

STRUCTURE

Feed-system architecture is an active design decision; pressure-fed is a plausible candidate.

RELATION

Helium pressurization is linked to maintaining feed pressure

PARAMETER

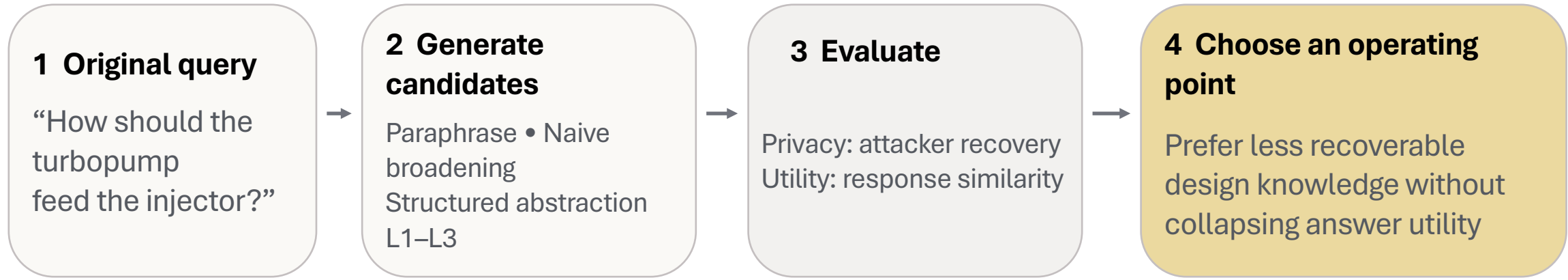
Likely chamber pressure $\approx$ 20 bar

RELATION

A regulated helium pressurization with feed pressure actively maintained throughout the burn

User-side protection: controlled semantic abstraction

The objective is not maximum hiding; it is a tunable privacy–utility operating point.



Structured abstraction changes what is explicit while preserving functional usefulness:

Original

turbopump → injector

Abstraction

pressure-raising component
→ downstream flow component

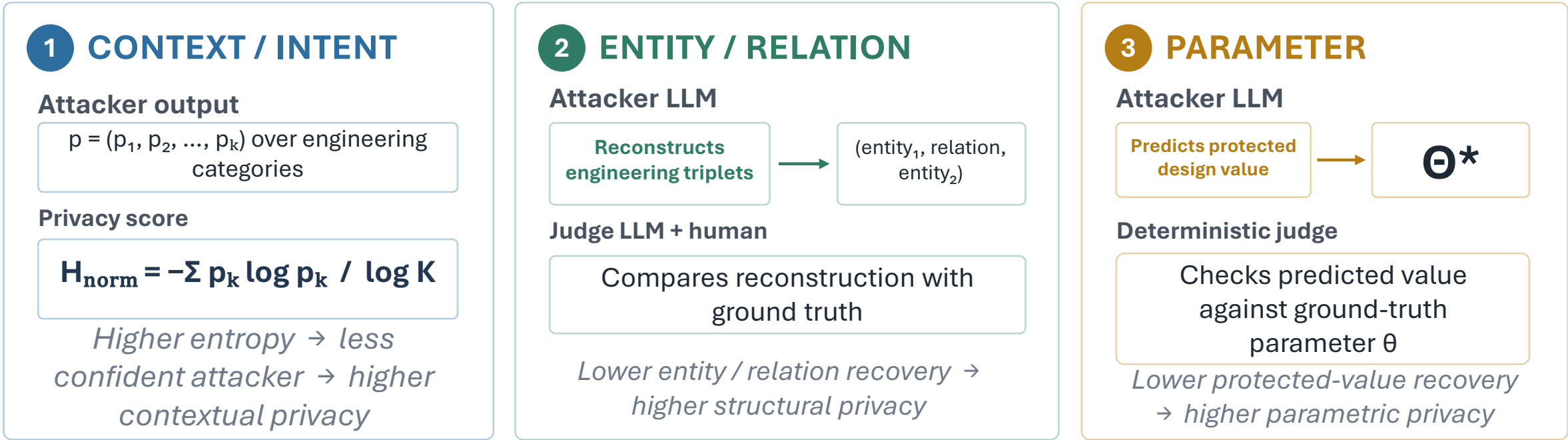
Research questions

- RQ1 What leaks from a single query?
- RQ2 What is the privacy–utility trade-off?
- RQ3 Does leakage differ by knowledge layer?
- RQ4 What accumulates across a query sequence?

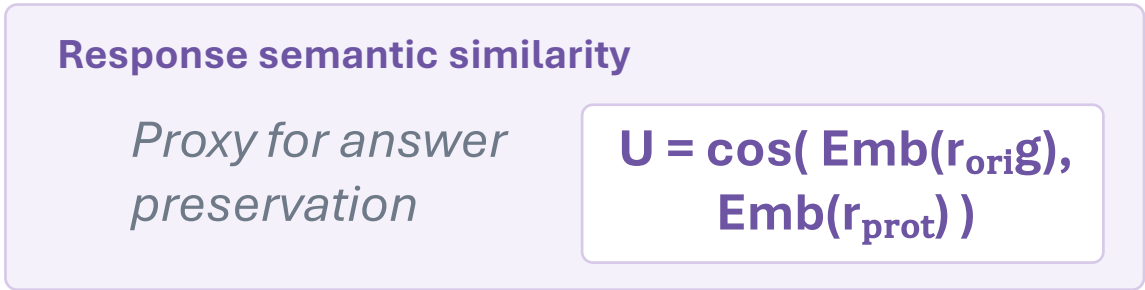
Controlled Evaluation: How Privacy and Utility Are Measured

Privacy is operationalized by knowledge layer; utility is measured independently as semantic response preservation.

LAYER-SPECIFIC PRIVACY EVALUATION



UTILITY

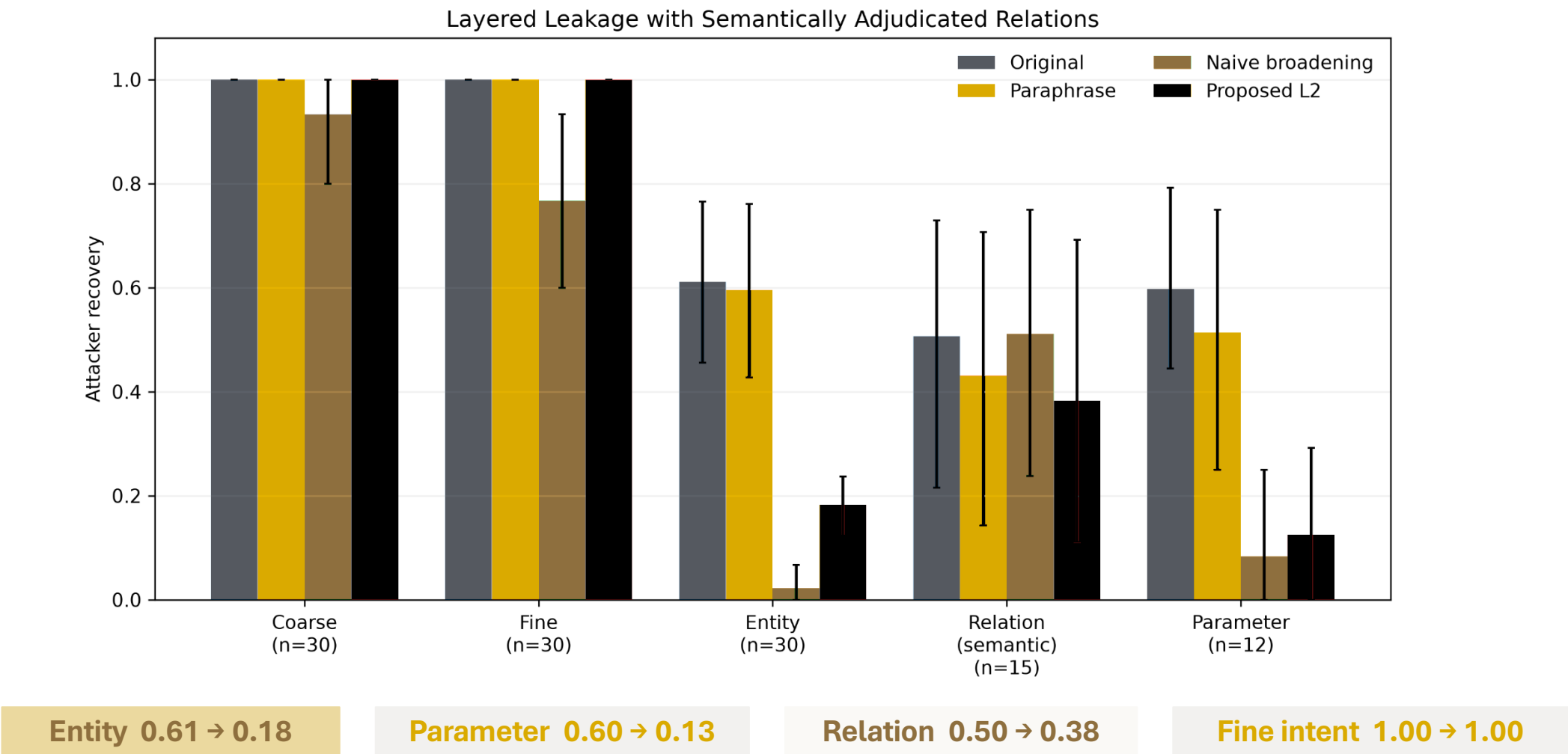


CONTROLLED DATASET

Dataset	Coverage	Methods	Samples
6 scenarios 30 queries 3 domains • 6 intents	Propulsion Structures GNC	Original Paraphrase Broadening L1-L3	Ctx/Ent 30 Rel 15 Param 12 Seq 6

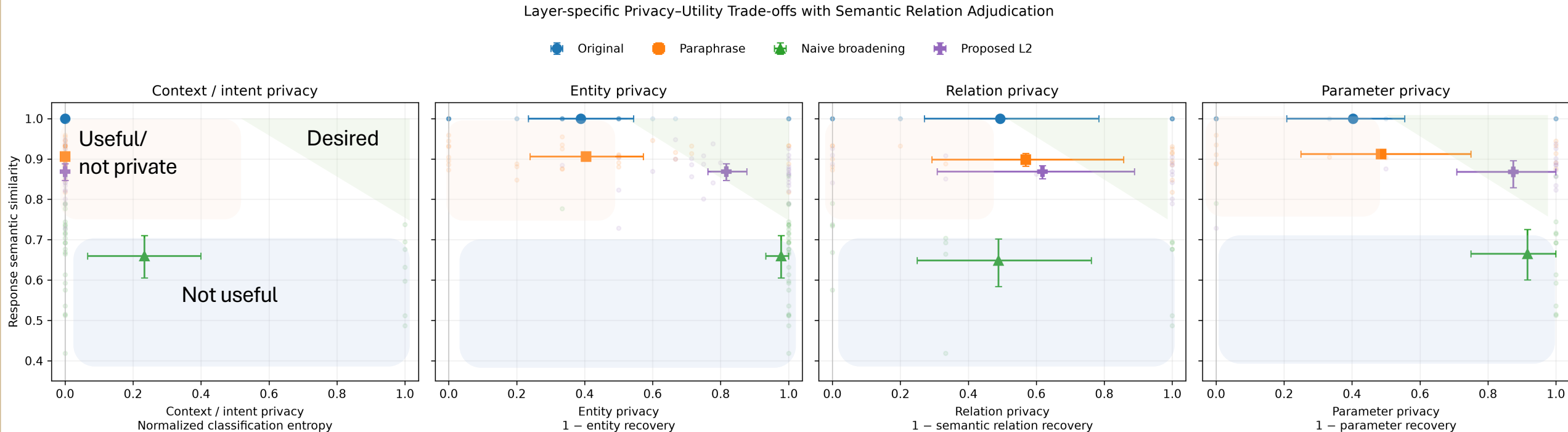
Result 1 — Abstraction hides explicit details more than latent intent or relations

Moderate abstraction (L2) reduces entity and parameter recovery, while context remains visible and relations remain resilient.



Result 2 — The “best” operating point depends on which privacy layer matters

A single privacy score would hide the fact that the same transformation behaves differently



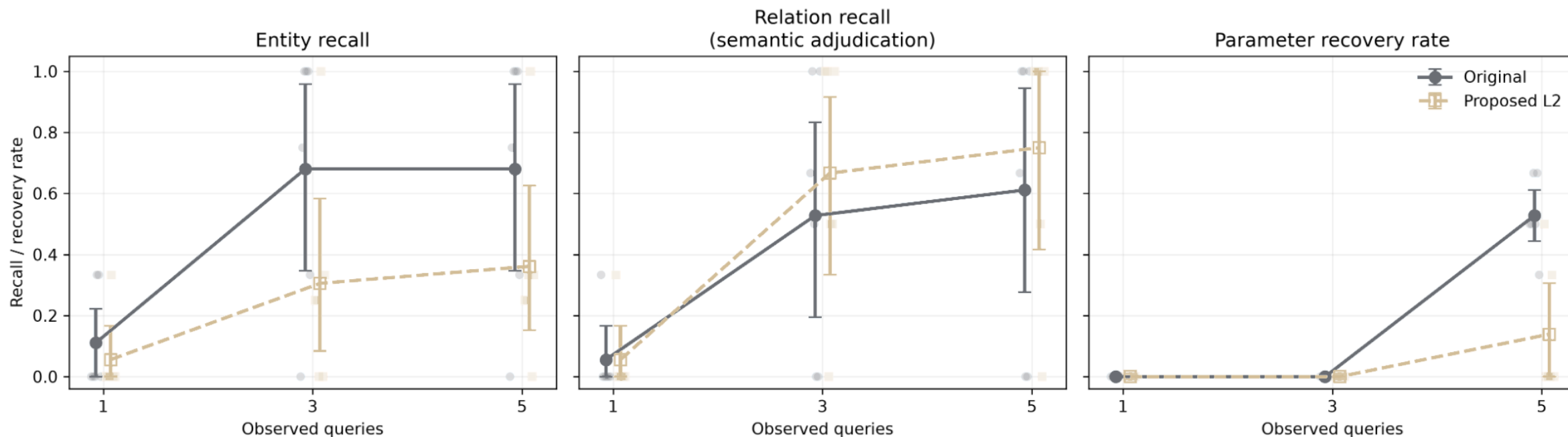
Observed pattern:

Moderate structured abstraction retains substantially higher response similarity while still suppressing entity and parameter leakage.

Result 3 — Sequential leakage is layer-dependent

Across six scenarios, L2 suppresses component and parameter recovery, but does not reduce functional-relation leakage.

Sequential Recall-focused Metrics (faint points: scenarios; lines: means with 95% CI)



Q1 Low structural / parameter recovery

Q1-3 Entity + relation recovery rises

Q1-5 Parameters emerge; relations persist

Core Insight: Engineering Privacy Is a Reconstruction Problem

The privacy risk is not only which facts are exposed, but how combined evidence narrows the hidden design space.

What the results imply

Explicit details are easier to hide

1

Entity and parameter leakage drops substantially under structured abstraction.

Latent structure remains visible

2

Fine intent and functional relations remain comparatively recoverable.

Interaction traces accumulate clues

3

Different queries expose complementary evidence about the same hidden design.

Why this matters at the system level

The same number of recovered facts can have very different privacy impact.

Before combining clues

Pressure-fed

Electric pump-fed

Turbopump-fed

combined
evidence

After combining clues

Pressure-fed **more likely**

Electric pump **less plausible**

Turbopump **less plausible**

Recovered facts become more privacy-sensitive when they collectively eliminate plausible system alternatives.

Takeaways

Three messages to carry forward from the controlled evaluation.

1

Engineering-query leakage is layered

Intent, structural knowledge, and parameters are not equally visible—or equally protectable.

2

Abstraction protects details better than latent structure

Entity and parameter recovery fall sharply, while relations and intent remain comparatively resilient.

3

Interaction history matters

Across related queries, different layers of protected design knowledge become recoverable at different stages.

**Engineering privacy should be evaluated by what an adversary can reconstruct
—not only by what words are removed.**



9/11/2026

Thanks!



School of Aeronautics
and Astronautics

Yinchien Huang

Research Assistant
PhD candidate

Center of Integrated System in Aerospace
Purdue University, ARMS office 3175
West Lafayette, IN 47906

Email: huan1903@purdue.edu
Mobile/WhatsApp: +1(765)251-7688



CENTER FOR INTEGRATED
SYSTEMS IN AEROSPACE

What Does Each Query-Protection Method Actually Change?

Same engineering need, different observable query forms — from wording-only changes to controlled semantic abstraction.

METHOD		EXAMPLE OBSERVABLE QUERY	WHAT REMAINS VISIBLE
BASELINE	Original	“How should the turbopump feed the injector?”	Explicit entities + functional relation Full design-specific wording
	Paraphrase	“What is the best way for the turbopump to deliver propellant to the injector?”	Same entities + same relation Wording changes; leakage largely stays
	Naive broadening	“How should one component supply another component in a propulsion system?”	Only broad system context Hides detail, but removes useful specificity
PROPOSED	Proposed L1	“How should the pumping component feed the downstream delivery component?”	Functional roles + relation Specific component names are abstracted
	Proposed L2	“How should the pressure-raising component supply the downstream flow component?”	Functional roles + topology cue More semantic abstraction; relation remains
	Proposed L3	“How should components in a pressurized fluid system be configured for downstream flow?”	System-level function Strongest abstraction of explicit identity

Increasing controlled abstraction strength

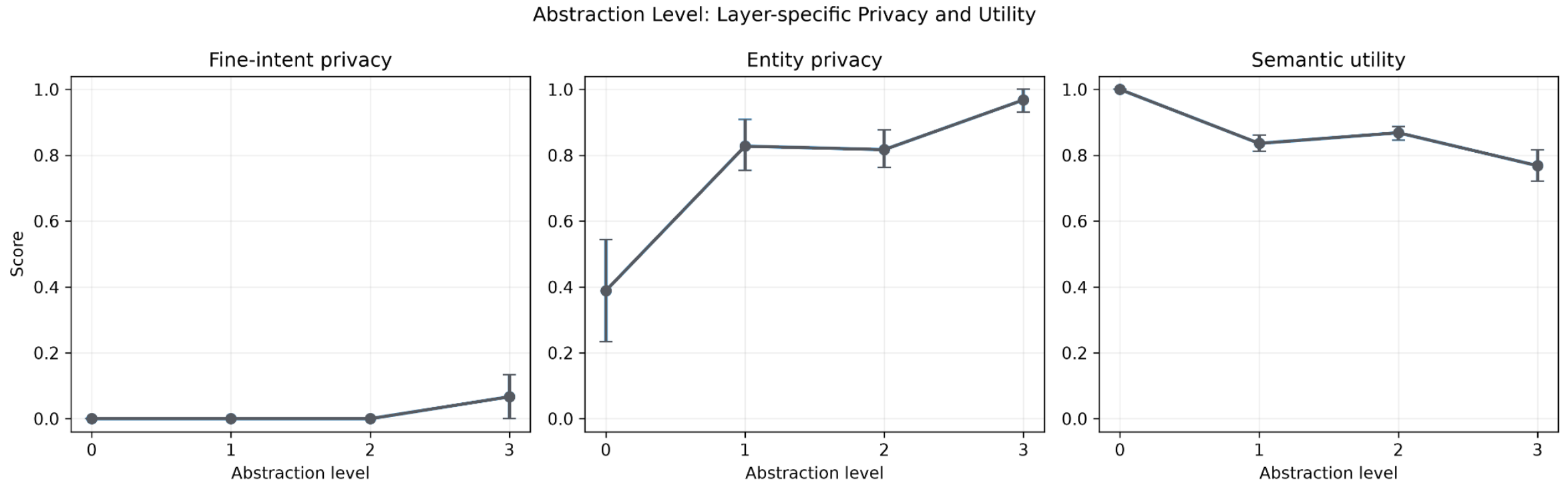
L1L2L3

Key idea Paraphrase changes wording; L1–L3 change what is explicit while trying to preserve functional usefulness.

L1–L3 are operational abstraction controls — not guaranteed monotonic privacy levels.

Backup — Abstraction-level behavior

Layer-specific response to increasing abstraction strength.



Note: the abstraction levels are not perfectly monotonic in utility or privacy; the result reinforces that “more abstract” is not universally better across layers.

Backup — Operational definitions

Representative indicators used for each knowledge layer.

Contextual

Coarse context recovery; fine-intent recovery

What engineering problem is being solved?

Structural

Entity recovery; semantically adjudicated relation F1

Which components and functional connections can be reconstructed?

Parametric

Protected-value recovery / tolerance-based recovery

Which numerical or categorical design values can be inferred?

Utility

Response embedding cosine similarity

Preliminary proxy for answer preservation—not engineering correctness

Semantic relation adjudication is necessary because exact lexical matching treats functionally equivalent relations (e.g., “feeds” vs. “supplies”) as different.