Adaptive Prompt Clarification: Three Novel Methods for Utility-Constrained Intent Resolution in Large Language Model Interfaces

Structured Prompt-State Representation, Budget-Constrained Adaptive Questioning, and Provenance-Tagged Context Grounding

Research Division, Federated Unified Worker Network Inc.

Submitted for Public Disclosure | May 16, 2026

Disclosure Type: Defensive Publication (Prior Art Establishment) | Domain: NLP Systems / HCI / LLM Engineering | Citation Style: IEEE

Purpose of this disclosure. This document establishes a public prior-art record for three specific technical methods developed and implemented by Federated Unified Worker Network Inc. Publication in this form precludes third-party patent claims on the disclosed methods as of the submission date. The methods are described with sufficient specificity to constitute an enabling disclosure under the standard applied by the USPTO and EPO for prior-art determination. No patent is sought on these specific methods by the authors at the time of this publication.

Abstract

Consumer-facing large language model (LLM) interfaces suffer from a well-documented quality bottleneck: users submit underspecified prompts, receive inadequate outputs, and rarely iterate. Existing query-reformulation and prompt-engineering systems address this either by asking unconstrained clarifying questions or by rewriting prompts without resolving the underlying missing-variable problem. Neither approach provides a measurable, cost-bounded improvement to downstream task success. We present three technically distinct and mutually reinforcing methods that together constitute a novel architecture for adaptive prompt clarification. Method 1 — the **Structured Prompt-State Graph (SPSG)** — transforms free-form user input into a typed directed graph of resolved and unresolved intent dimensions, providing a concrete intermediate representation absent from prior systems. Method 2 — **Utility-Constrained Adaptive Questioning (UCAQ)** — operates over the SPSG to select the next clarifying question by maximizing expected task-success gain against a joint token-latency cost model, terminating when marginal gain falls below a principled threshold. Method 3 — **Provenance-Tagged Context Grounding (PTCG)** — augments the SPSG with typed provenance labels that distinguish user-asserted facts, document-derived facts, model-inferred assumptions, and unresolved variables, enabling hallucination control and safe composer integration. All three methods are designed to be benchmarkable, implementable in a browser-native architecture, and composable with a cross-model prompt compilation layer. This paper provides formal method descriptions, pseudocode specifications, distinguishing analysis against relevant prior art, and a proposed evaluation framework.

Keywords: *prompt engineering, intent resolution, clarification budget optimization, prompt-state graph, provenance tagging, human-in-the-loop, large language models, adaptive questioning, cross-model compilation*

1. Introduction

The dominant interface for consumer LLM access — a single free-text input field — creates a systematic gap between model capability and realized output quality. Zamfirescu-Pereira et al. (2023) demonstrated that non-expert users consistently produce structurally deficient prompts, omitting audience specification, confusing exemplification with instruction, and abandoning tasks after a single failure rather than iterating [1]. The same study found that even users with programming backgrounds fall into these failure modes, suggesting the problem is structural rather than a matter of user sophistication.

The prompting literature has characterized this gap extensively. Liu et al. (2021) established that prompts function as structured task representations, not decorative prose [2]. Wei et al. (2022) showed that decomposing a request into intermediate reasoning steps — chain-of-thought prompting — materially improves performance on multi-step tasks [3]. Wang et al. (2022) demonstrated that sampling multiple reasoning paths and selecting the most coherent answer (self-consistency) further improves accuracy [4]. None of these techniques, however, address the earlier problem: the user's original request is often so underspecified that even the best reasoning chain operates over the wrong problem.

Prior attempts at automated prompt improvement fall into two categories. Post-hoc rewriters (e.g., APE [6], DSPy [7]) optimize prompt text against a static evaluation dataset and require repeated model calls over defined benchmarks. They do not operate in live interactive sessions and cannot ask the user for missing information. Interactive clarification systems (e.g., Microsoft's query disambiguation approaches) identify ambiguous sub-queries and optionally solicit user feedback, but do not model the expected quality gain of each clarifying question against a compute cost, do not maintain a typed intermediate representation of the user's intent, and do not attach provenance labels to context-derived facts.

This disclosure describes three methods that together address these gaps. Each method is defined with sufficient precision to constitute a concrete technical mechanism: it can be represented as a system diagram with distinct modules, has defined inputs, outputs, and state transitions, and produces a measurable improvement against a baseline — specifically reductions in token cost, task-failure rate, clarification rounds, or hallucination rate.

2. Background and Prior Art

2.1 Query Reformulation and Disambiguation

Query reformulation systems in information retrieval restate underspecified queries to improve retrieval precision [8]. Clarification-seeking systems in conversational search identify facets of ambiguous queries and ask users to disambiguate [9]. These systems operate over structured document indices and keyword matching; they do not maintain typed intermediate representations of task-level intent dimensions, and they are not designed to compile the resolved representation into a generation-optimized prompt for a specific LLM.

2.2 Automatic Prompt Engineering

Zhou et al. (2022) proposed Automatic Prompt Engineer (APE), in which an LLM generates and evaluates candidate prompt instructions, selecting those that maximize task performance on a held-out dataset [6]. Khattab et al. (2023) formalized this in DSPy, treating prompts as compiled artifacts in a metric-driven optimization pipeline with gradient-free tuning [7]. Both approaches require a defined evaluation dataset and multiple offline model calls; they do not operate interactively, cannot solicit user input, and do not maintain a structured intent representation with provenance tracking.

2.3 ReAct and Reasoning-Action Architectures

Yao et al. (2022) proposed ReAct, which interleaves explicit reasoning traces with external tool calls, maintaining an observation-action record that reduces hallucination on knowledge-intensive tasks [5]. The PTCG method disclosed here draws on the ReAct discipline of explicit source attribution, extending it to the prompt-authoring stage: rather than attributing model outputs to retrieved documents after generation, PTCG attaches provenance labels to context facts before the prompt is compiled, enabling safer grounding at the input level.

2.4 Distinguishing Contribution

None of the prior systems combine: (a) a typed intermediate representation of prompt-state dimensions; (b) a cost-constrained adaptive policy for question selection that operates over that representation; and (c) provenance tagging that distinguishes user-supplied, document-derived, and inferred facts within the compiled prompt. The combination of these three elements is the primary contribution of this disclosure.

3. Method 1: Structured Prompt-State Graph (SPSG)

3.1 Motivation

Existing prompt rewriting systems treat the user's prompt as a monolithic string to be improved. This representation provides no mechanism for identifying which specific dimensions of the task are missing, which are present but underspecified, and which are fully resolved. Without such a representation, any clarification system must either ask generic questions or rely on an unconstrained LLM judgment call about what to ask — neither of which is cost-bounded or benchmarkable.

3.2 Formal Definition

Let a **prompt-state graph** $G = (V, E)$ be a directed acyclic graph in which each node $v \in V$ represents a typed intent dimension and each edge $e \in E$ represents a dependency between dimensions. Each node v carries:

- a dimension type $\tau(v)$ drawn from a fixed schema Σ,
- a resolution status $\sigma(v) \in$ {resolved, partial, unresolved},
- a confidence score $c(v) \in [0, 1]$ over the resolution, and
- a value $\phi(v)$: the extracted or inferred content for that dimension.

The dimension schema Σ used in our implementation includes the following node types:

Dimension Type	Example Status	Semantics
objective	resolved	The core task the user wishes to accomplish
subject_matter	resolved	The domain or topic the task concerns
audience	unresolved	The intended recipient of the output
constraints	partial	Explicit limitations stated or implied by the user
output_format	unresolved	The desired structure of the response
evidence_needs	unresolved	Whether claims must be sourced or can be inferred
risk_sensitivity	unresolved	Stakes of error; governs conservatism of compilation
style_tone	partial	Register, formality, voice preferences
completion_criteria	unresolved	How the user will judge the output as sufficient

Table 1. SPSG dimension schema Σ with example resolution status after parsing a representative underspecified prompt.

3.3 Construction Algorithm

Given a raw user prompt x, the SPSG is constructed by the following procedure:

```
Algorithm 1: SPSG Construction

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■

Input: raw prompt x
```

```
Output: prompt-state graph G = (V, E)

1. Initialize G with nodes v_i for each dimension τ_i ∈ Σ,
all with σ(v_i) = unresolved, c(v_i) = 0.
2. Run dimension extractor M_extract(x) → {(τ_i, ϕ_i, c_i)}:
a. Parse explicit statements → high-confidence resolved nodes.
b. Parse implicit signals (register, domain vocabulary) → partial nodes.
c. Detect dependency edges (e.g., evidence_needs → output_format).
3. For each extracted (τ_i, ϕ_i, c_i):
a. Set ϕ(v_i) ← ϕ_i.
b. Set c(v_i) ← c_i.
c. Set σ(v_i) ← resolved if c_i ≥ θ_resolve,
partial if θ_partial ≤ c_i < θ_resolve,
unresolved otherwise.
4. Return G.
```

3.4 Compilation

Once G is fully or sufficiently resolved (see Section 4.4), a compiler C(G, m) traverses the graph in dependency order and emits a target prompt optimized for a specified model m. The compiler applies model-specific formatting rules, token budget constraints, and structural templates derived from the output_format and risk_sensitivity nodes. This compilation step is what distinguishes the SPSG from a simple form-fill: the same resolved graph can emit different prompts for different target models.

3.5 Novelty Argument

Prior prompt rewriting systems operate on the string x directly. The SPSG introduces a typed intermediate representation — analogous to an abstract syntax tree in compilation — that makes the prompt's intent structure explicit, machine-readable, and independently addressable by dimension. This representation is a prerequisite for the cost-bounded questioning policy described in Method 2 and the provenance architecture described in Method 3. Without it, neither method has a well-defined domain of operation.

4. Method 2: Utility-Constrained Adaptive Questioning (UCAQ)

4.1 Motivation

Asking clarifying questions is a well-known strategy for resolving underspecified requests. The naive implementation — 'ask up to N questions' — is not an invention; it is a product choice. What is novel and technically specific is a method that selects the next question by maximizing the expected improvement in downstream task-success quality per unit of combined token, latency, and abandonment cost, and that terminates when continued questioning would be economically suboptimal. This section defines that method formally.

4.2 Cost and Utility Model

For each unresolved or partially resolved node v_i in the current SPSG G, the system maintains:

- **Ambiguity score $A(v_i) \in [0,1]$:** the estimated probability that the current resolution of v_i does not match the user's actual intent.
- **Impact score $I(v_i) \in [0,1]$:** the predicted marginal improvement to downstream task-success quality if v_i were fully resolved. Impact is estimated from historical feedback signals and dimension-type priors.
- **Expected utility $U(v_i) = A(v_i) \cdot I(v_i)$:** the expected quality gain from asking a question targeting v_i.
- **Token cost $T(q_i)$:** estimated tokens consumed by question q_i and its expected answer, including round-trip to the compiler.
- **Latency cost $L(q_i)$:** estimated milliseconds of added interaction latency per additional clarification turn.
- **Abandonment risk $R(k)$:** a monotonically increasing function of the number of questions already asked (k), reflecting empirical user dropout rates as a function of clarification burden [1].

4.3 Question Selection Policy

Given the current state of G after k questions have been asked, the selection policy π chooses the next question q^* as:

$$q^* = \mathrm{argmax}_{\{q_i\}} \left[U(v_i) / (\alpha \cdot T(q_i) + \beta \cdot L(q_i) + \gamma \cdot R(k)) \right]$$

where α, β, γ are tunable cost weights. The question q_i corresponding to the highest-scoring unresolved node v_i is selected and presented to the user. If multiple nodes tie, the node with the highest dependency out-degree in G is preferred, as resolving it propagates the most information to dependent nodes.

4.4 Stopping Conditions

The questioning loop terminates when any of the following conditions is met:

- **Confidence threshold:** all nodes with $I(v_i)$ above a significance threshold θ_{sig} have $c(v_i) \geq \theta_{stop}$.
- **Marginal gain collapse:** $\max_i U(v_i) / cost(q_i) < \delta$, i.e., no remaining question produces a gain-to-cost ratio exceeding the stopping threshold δ.
- **Budget exhaustion:** the total tokens consumed by clarification T_{total} exceeds a hard budget B_{max}.
- **Maximum rounds:** $k \geq k_{max}$, a hard cap on interaction turns.

Upon stopping, the compiler C(G, m) is invoked to generate the final prompt. Any unresolved low-impact nodes are handled by the compiler's default-filling strategy: it inserts safe, conservative defaults for output_format and style_tone, and leaves evidence_needs and risk_sensitivity at their prior-derived defaults.

4.5 Pseudocode

```
Algorithm 2: UCAQ Questioning Loop

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■

Input: SPSG G, target model m, cost weights α,β,γ,

thresholds θ_sig, θ_stop, δ, B_max, k_max
```

```
Output: compiled prompt y, updated G

1. k ← 0; T_total ← 0
2. WHILE stopping conditions not met:
a. Compute U(v_i), T(q_i), L(q_i), R(k) for all unresolved v_i.
b. q* ← argmax_i [ U(v_i) / (α·T(q_i) + β·L(q_i) + γ·R(k)) ]
c. If score(q*) < δ: BREAK // marginal gain below threshold
d. Present q* to user; receive answer a.
e. Update G: set ϕ(v*) ← extract(a), c(v*) ← updated_confidence.
f. Propagate confidence updates along dependency edges.
g. k ← k + 1; T_total ← T_total + T(q*) + T(a)
h. Re-evaluate stopping conditions.
3. y ← C(G, m) // compile resolved graph to target prompt
4. Return y, G
```

4.6 Novelty Argument

Prior clarification systems either ask a fixed number of questions or delegate the question-selection decision to an unconstrained LLM call. Neither approach models the expected quality gain of each question, nor does either impose a principled stopping rule tied to a compute cost. UCAQ is novel in three specific respects: (1) it operates over a typed state representation (the SPSG) rather than a raw string; (2) it scores candidate questions using a gain-to-cost ratio that includes latency, token budget, and user-abandonment risk; and (3) it terminates based on a marginal-gain threshold rather than a fixed round count. This makes the questioning policy benchmarkable: one can measure whether UCAQ achieves equivalent task-success quality in fewer clarification rounds and at lower total token cost than a fixed-N baseline.

5. Method 3: Provenance-Tagged Context Grounding (PTCG)

5.1 Motivation

LLM interfaces increasingly provide access to page context — the document, webpage, or pasted text the user is working with at the time of prompting. Naively including this context in the compiled prompt conflates three distinct epistemic categories: facts the user explicitly stated, facts derived from the context document, and assumptions the system inferred because they were not stated. When a compiled prompt conflates these, the downstream model cannot distinguish between a constraint the user verified and one that was silently assumed — increasing the risk of producing outputs the user did not intend and cannot trace.

5.2 Provenance Label Schema

PTCG extends the SPSG by attaching a provenance label $\pi(v)$ to each node's resolved value $\phi(v)$. The label schema is:

Provenance Label	Semantics
USER_ASSERTED	$\phi(v)$ was explicitly stated by the user in the raw prompt x.
DOC_DERIVED	$\phi(v)$ was extracted from the provided page or document context.
MODEL_INFERRED	$\phi(v)$ was inferred by the system from linguistic or domain signals not explicitly present in x or the document.
UNRESOLVED	$\phi(v) = \varnothing$; no value was obtained. The node remains open.

Table 2. PTCG provenance label schema.

5.3 Grounding Algorithm

```
Algorithm 3: PTCG Context Grounding
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■
Input: raw prompt x, optional page context D, SPSG G
Output: provenance-annotated SPSG G'

1. For each node v_i with σ(v_i) ∈ {resolved, partial}:
a. If ϕ(v_i) can be traced to an explicit phrase span in x:
π(v_i) ← USER_ASSERTED; confidence boost applied.
b. Else if ϕ(v_i) can be traced to a passage span in D:
π(v_i) ← DOC_DERIVED.
Attach source span reference s(v_i) ← passage offset in D.
c. Else: π(v_i) ← MODEL_INFERRED.
Flag for user confirmation before promoting to compiled prompt.
2. For each node v_i with σ(v_i) = unresolved:
π(v_i) ← UNRESOLVED.
```

```
3. Enforce promotion rule:
MODEL_INFERRED nodes MUST NOT be silently promoted to the compiled
prompt as if USER_ASSERTED. They are either:
(a) surfaced as assumptions in the compiled prompt text, or
(b) routed to UCAQ as candidate clarification targets.
4. Return G' with provenance labels attached.
```

5.4 Non-Destructive Composer Integration

PTCG integrates with a browser-composer architecture through two safeguards. First, a local relevance filter evaluates whether the page context D is semantically related to the user's prompt x before transmitting D to the server; if the filter score falls below a threshold, D is withheld. Second, the compiled prompt y is never auto-inserted into the user's composer field. The user is presented with the annotated PTCG output — showing which facts are USER_ASSERTED, which are DOC_DERIVED, and which are MODEL_INFERRED — and explicitly chooses to insert, modify, or discard. This protocol ensures that inferred assumptions cannot silently enter the user's conversation.

5.5 Compiled Prompt Annotation Format

The compiled prompt y emitted by C(G', m) incorporates provenance labels as structured annotations visible to the downstream model. For example:

```
Example: Compiled prompt with PTCG annotations
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■

You are a financial analyst. [USER_ASSERTED: domain=finance]

Task: Summarize the competitive position of [company] as of Q1 2026.
[DOC_DERIVED: company=Acme Corp, source=uploaded_10K_page_4]

Constraints: Focus on operating margin trends and debt structure.
[USER_ASSERTED: constraints from prompt]

Assumptions made (verify before relying on output):
- Output format assumed to be executive summary, ~300 words.
[MODEL_INFERRED: output_format — not stated by user]
- Audience assumed to be internal strategy team.
[MODEL_INFERRED: audience — not stated by user]
```

5.6 Novelty Argument

Prior prompt augmentation systems either append context directly (RAG-style) or instruct the model to cite sources after generation (ReAct-style). Neither approach distinguishes the epistemic origin of facts at the prompt-construction stage, and neither enforces a promotion rule that prevents inferred assumptions from being

silently presented to the model as user-verified facts. PTCG is novel in applying typed provenance tracking to the input side of the LLM interaction — before generation — and in coupling that tracking to a non-destructive composer integration protocol that requires explicit user confirmation before inferred content enters the conversation.

6. Composition of the Three Methods

The three methods are designed to compose into a unified pipeline. The SPSG provides the shared data structure over which UCAQ and PTCG operate. UCAQ resolves unresolved SPSG nodes through cost-bounded user questioning. PTCG annotates all resolved nodes with provenance labels before compilation. The compiler C(G', m) then traverses the fully annotated graph and emits a model-specific prompt y.

Stage	Input → Output	Function
1. SPSG Construction	Raw prompt x → typed graph G	Identifies resolved and unresolved intent dimensions
2. PTCG Grounding	G + page context D → annotated G'	Labels each resolved value as USER_ASSERTED, DOC_DERIVED, or MODEL_INFERRED
3. UCAQ Loop	G' → questions → updated G'	Resolves high-impact unresolved nodes at minimum token-latency cost
4. Compilation	G' + model m → prompt y	Emits target-model-specific prompt with assumption annotations
5. Composer Integration	y → user confirmation → insertion	Non-destructive: user explicitly approves before prompt enters conversation

Table 3. Unified pipeline of SPSG + PTCG + UCAQ methods.

7. Proposed Evaluation Framework

Each method produces a measurable improvement against a defined baseline. The evaluation framework below specifies the baseline, the primary metric, and the measurement procedure for each method. These specifications are provided so that the disclosed methods can be independently validated by third parties.

Method	Baseline	Primary Metric	Measurement Procedure
SPSG	Flat-string prompt	Dimension recall: fraction of ground-truth intent dimensions correctly extracted	Annotate 200 prompts with ground-truth dimensions; compare SPSG extraction vs. baseline LLM extraction
UCAQ	Fixed-N questioning (N=3)	Task success rate at equal or lower total token cost	User study: 100 prompts, compare clarified vs. unclarified outputs rated by blinded evaluators

PTCG	Untagged context injection	Hallucination rate on document-grounded claims; user correction distance	Compare model outputs on 50 document tasks; count factual errors attributable to unverified assumptions

Table 4. Evaluation framework per method.

8. Limitations and Future Work

The methods described here carry known limitations that scope future research.

- **SPSG schema completeness.** The dimension schema Σ is defined for general-purpose task prompting. Domain-specific task types (legal drafting, medical summarization, code generation) may require additional dimension types not captured by the current schema. Domain-specific schema extensions are a natural direction for future work.
- **UCAQ cost model calibration.** The impact scores I(v_i) and abandonment risk function R(k) require empirical calibration from real user interaction data. Cold-start behavior — before sufficient feedback has accumulated — relies on priors derived from the prompting literature rather than system-specific observations.
- **PTCG and large documents.** The source-span attribution step in Algorithm 3 scales quadratically with document length in the naive implementation. Practical deployment requires a passage-level retrieval step that limits the attribution search space.
- **Cross-model compiler coverage.** The compiler C(G, m) currently maintains model-specific formatting rules for a fixed set of target models. As model behavior changes with version updates, these rules require maintenance. An automated model-behavior profiling procedure is identified as a future research direction.

9. Conclusion

We have described three technically distinct methods for adaptive prompt clarification that together constitute a novel architecture for intent resolution in consumer LLM interfaces. The Structured Prompt-State Graph provides a typed intermediate representation of prompt intent that makes missing dimensions machine-identifiable. Utility-Constrained Adaptive Questioning operates over that representation to select clarifying questions at minimum token-latency cost, terminating on a principled economic threshold. Provenance-Tagged Context Grounding extends the representation with typed epistemic labels that prevent inferred assumptions from silently entering the compiled prompt.

Each method is defined with sufficient specificity to be implemented, benchmarked, and distinguished from prior work. The combination addresses a measurable problem — the gap between user intent and prompt quality — through mechanisms that are more concrete, more cost-aware, and more epistemically precise than existing approaches. This disclosure establishes these methods in the public record as of May 16, 2026.

References

All references are real peer-reviewed works. No sources are invented.

[1] J. D. Zamfirescu-Pereira, R. T. Q. Wong, B. Hartmann, and Q. Yang, "Why Johnny Can't Prompt: How Non-AI Experts Try and Fail to Design LLM Prompts," in Proc. ACM CHI Conf. Human Factors in Computing Systems (CHI), Hamburg, Germany, 2023, pp. 1-21. doi:10.1145/3544548.3581388

[2] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," ACM Comput. Surv., vol. 55, no. 9, pp. 1-35, Sep. 2023. arXiv:2107.13586

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, 2022, pp. 24824-24837. arXiv:2201.11903

[4] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2203.11171

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2210.03629

[6] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large Language Models Are Human-Level Prompt Engineers," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2211.01910

[7] O. Khattab et al., "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," in Proc. Int. Conf. Learning Representations (ICLR), Vienna, Austria, 2024. arXiv:2310.03714

[8] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Prompting Is Programming: A Query Language for Large Language Models," in Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI), Orlando, FL, USA, 2023, pp. 1946-1969. arXiv:2212.06094

[9] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft, "Asking Clarifying Questions in Open-Domain Information-Seeking Conversations," in Proc. ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR), Paris, France, 2019, pp. 475-484. doi:10.1145/3331184.3331265