RESEARCH ARTICLE OPEN

LLMs as Database Administrators: A Survey of AI-Driven Schema Design and Index Recommendation

Mohamed Chetouani*, Mahfoudi Souhail **

* Nanjing University of Information Science and Technology

Email: medchet2015@gmail.com

**Nanjing University of Information Science and Technology

Email: souhail.mah@gmail.com

----------**********----------

Abstract:

Large language models are reshaping database administration by enabling automation across tasks — schema design, index recommendation, configuration tuning, and diagnosis — that classical cost-model-driven tools handle only narrowly. This survey covers ten representative systems published between 2023 and 2026, organized under a four-axis taxonomy of task scope, LLM integration paradigm, deployment model, and autonomy level, with deep-dive comparisons on the two most mature tasks: index recommendation and schema design. For index recommendation, LLM-based advisors such as LLMIA, LLMIdxAdvis, and MAAdvisor match or exceed production baselines like Microsoft's DTA, though a persistent gap between recommendation quality and validation cost remains unresolved. For schema design, the literature is earlier-stage and lacks shared benchmarks. Across all systems, three findings recur: database feedback loops separate effective advisors from naive prompting baselines, hallucination takes domain-specific forms requiring targeted mitigation, and the tension between frontier-model capability and on-premise deployment constraints is unresolved. Five open challenges — schema scale, cost-model coupling, workload drift, trust and explainability, and standardized benchmarking — define the road ahead for LLM-driven database administration.

Keywords **— LLMs, database administration, index recommendation, schema design, in-context learning, retrieval-augmented generation, multi-agent systems, database tuning, DBA automation, large language models,**

----------**********----------

I. INTRODUCTION

A. The DBA role and the automation gap

Database administrators sit at the intersection of workload understanding, physical design, and operational recovery. Their responsibilities span schema design, index and view selection, configuration tuning, query-performance troubleshooting, capacity planning, backup and replication, and post-incident diagnosis. Each of these tasks is knowledge-intensive: good decisions depend not only on the current workload and schema but also on accumulated experience with the engine's quirks, with historical anomaly patterns, and with organization-specific conventions that rarely appear in any manual. In cloud settings, this cognitive load scales poorly. A single team may be nominally responsible for millions of database instances, a ratio at which manual diagnosis is acknowledged to be infeasible and existing rule-based or small-model tooling is explicitly characterized as "one small piece of a bigger puzzle" [1]. The gap between the volume of

DBA work that needs doing and the number of experts available to do it is the structural problem that motivates automation research.

Classical database automation has made that gap narrower but has not closed it. Cost-based advisors such as Microsoft's Database Tuning Advisor (DTA) and AutoAdmin formulate index selection as a constrained search over hypothetical configurations, using the optimizer's what-if interface to estimate benefit without materializing candidates [1, 7]. Learning-based successors cast the same problem as reinforcement learning or Bayesian optimization, iteratively refining configurations through benefit evaluations [2]. Both families share two limitations that matter for this survey. First, they operate on narrow, well-formalized tasks and do not generalize across the full DBA workflow: an index advisor cannot diagnose a replication lag, and a knob tuner cannot read a support ticket. Second, they depend on cost-model fidelity. Recent work has shown that the correlation between optimizer-estimated cost and actual query latency is weak enough to lead cost-guided search to measurably sub-optimal recommendations [2], and that the validation overhead of building recommended indexes can exceed the tuning cost itself [3].

B. Why LLMs change the picture

Large language models alter both constraints. On the breadth side, a single pretrained model can be prompted across tasks that previously required separate, specialized pipelines: reading a tuning manual, parsing an error log, drafting a schema, explaining an execution plan, or generating a SQL rewrite. The same backbone that powers natural-language interfaces to databases [3] can also be assigned the role of a diagnostic expert that retrieves knowledge from documents, invokes database tools, and produces a reasoned report [5]. On the depth side, LLMs offer a substrate for combining capabilities that classical systems kept separate. In-context learning supplies task specialization without fine-tuning; retrieval-augmented generation (RAG) injects private or proprietary knowledge at inference time; tool use lets the model call EXPLAIN, a what-if caller, or an external cost model; and multi-agent orchestration decomposes complex workflows into interacting roles with distinct specializations. Recent LLM-driven DBA systems draw on some subset of these capabilities: D-Bot assembles diagnosis experts with tree-search reasoning and document-grounded knowledge [9]; λ-Tune generates whole configurations rather than individual knob suggestions, avoiding the combinatorial trial loop of earlier learned tuners [9]; MAAdvisor decomposes index selection into planning, selection, combination, revision, and reflection agents [10]; and DB-GPT integrates fine-tuned text-to-SQL models with privacy-preserving local deployment [3].

The change, however, is not only a matter of capability. It introduces new failure modes that classical advisors do not exhibit. LLMs hallucinate indexes on non-existent columns, overproduce recommendations when under-constrained, and remain sensitive to prompt structure in ways that complicate reproducibility. Empirical head-to-head studies against DTA have already documented that straightforward LLM-based index tuners, while competitive, do not uniformly dominate industrial cost-based baselines on real customer workloads [7], and that their recommendations exhibit high variance across invocations. A realistic account of LLM-driven database administration therefore has to hold two claims in tension: LLMs enable tasks that were previously infeasible, and they do so while inheriting calibration, reliability, and verification problems that the DBA community has spent decades engineering around.

C. Scope, contributions, and paper organization

This survey covers work published between 2023 and 2026 that applies large language models to tasks traditionally owned by database administrators. The coverage is deliberately broad, spanning holistic copilots, system tuning, diagnosis, data profiling, and natural-language-to-database interfaces, but the anchoring depth is placed on two tasks: index recommendation and schema design. These are the two problems where classical

automation is most mature, where the LLM literature is most concentrated, and where the contrast between paradigms is sharpest. We treat adjacent areas such as text-to-SQL and learned query optimization only insofar as they inform the DBA-facing tasks under study; comprehensive treatments of those areas exist elsewhere [7, 9].

The survey makes three contributions. First, it organizes the recent literature under a compact taxonomy along four axes: task scope, LLM integration paradigm, deployment model, and autonomy level. Second, it provides side-by-side treatments of index recommendation and schema design that go beyond description to compare how different systems resolve the same design tensions: grounding through retrieval versus through tool feedback, single-shot versus iterative refinement, and single-agent versus multi-agent decomposition. Third, it extracts cross-cutting patterns and open problems that no individual paper surfaces on its own: the recurring tension between cost-model dependence and actual-latency validation, the underdeveloped state of benchmarking for LLM-driven DBA tasks, and the trust and verification gap that currently prevents recommendations from flowing directly into production without human review.

II. BACKGROUND

A. Core DBA tasks and classical automation

The work of a database administrator decomposes into a loosely coupled set of tasks that share a dependence on workload characteristics and engine internals but otherwise admit very different formalizations. Physical design — choosing indexes, materialized views, partitioning strategies, and clustered keys — is cast as a constrained optimization over a combinatorial configuration space. Configuration tuning sets values for tens to hundreds of engine knobs that interact non-linearly with workload and hardware. Schema design translates domain requirements into normalized relational structures. Diagnosis inspects metrics, logs, and plans to explain deviations from expected behavior. And query-level assistance covers rewriting, plan inspection, and SQL authoring. Most of these tasks are knowledge-intensive in ways that classical optimization frameworks capture only partially: a good configuration depends not just on the current workload but on engine-specific folklore, on organizational conventions, and on accumulated patterns of what tends to go wrong.

Index recommendation is the task where classical automation is most mature and where its mechanics most clearly frame the rest of this survey. The problem is NP-complete, which rules out exact search, so advisors decompose it into three stages: candidate generation over indexable columns, index selection under a storage or cardinality budget, and benefit evaluation that scores candidates against the workload [7]. The two dominant families — heuristic search and learning-based search — differ in how they explore the selection space but share a common dependence on cheap benefit estimation. Heuristic advisors such as AutoAdmin, Microsoft's DTA, DB2Advis, Relaxation, Drop, and Extend apply greedy, incremental, or relaxation-based search over candidate configurations [1, 3]. Learning-based successors reformulate selection as a sequential decision problem: AutoIndex uses Monte Carlo tree search, DQN and SWIRL apply Deep Q-Networks and proximal policy optimization, BALANCE adds a transfer mechanism for workload drift, and MFIX uses Bayesian optimization [5, 7]. Configuration tuning follows a parallel trajectory, with systems such as OtterTune, LlamaTune, and ParamTree using machine learning to navigate the knob space under trial-based feedback [1].

Both families depend on the same enabling primitive: the optimizer's what-if interface, which materializes hypothetical indexes virtually and returns an EXPLAIN-style estimated cost without building them on disk [3]. Without what-if estimation, the benefit-evaluation loop would require physically creating candidate indexes and running the workload, which is prohibitive at the scale of hundreds or thousands of evaluations. Heuristic advisors have been observed to issue over 20,000 benefit evaluations on a single TPC-DS tuning run, and learning-based

methods require 100 to 400 iterations each involving at least one such evaluation [7]. The efficiency of the entire classical pipeline rests on this primitive.

This reliance is also the primary weakness the LLM-based literature is reacting to. Optimizer cost estimates correlate imperfectly with actual query latency, so configurations that minimize estimated cost do not always minimize measured latency [7]. Empirical studies show heuristic advisors producing regressions on specific query templates even when overall workload cost improves [7], and independent benchmarking against DTA has documented that validating recommended configurations by building the indexes and running the workload can cost more than the tuning itself [7]. Classical automation therefore confronts a structural limitation that motivates much of the work surveyed in Sections 4 through 6: the tooling is efficient because it trusts the cost model, and the cost model is demonstrably imperfect.

B. LLM capabilities relevant to DBA work

Four capabilities of large language models shape how they are integrated into DBA systems, and every system in this survey can be located on the axes they define.

In-context learning (ICL) supplies task specialization without fine-tuning. A prompt containing a small number of demonstrations — input-output pairs drawn from a curated pool — steers a pretrained model toward a target task at inference time [1]. The advantages are operational: no gradient updates, no held-out training set, and adaptation to new schemas or workloads by swapping demonstrations rather than retraining. The costs are equally concrete. ICL is sensitive to demonstration choice and ordering, zero-shot prompting exhibits limited effectiveness on database-specific tasks because general-purpose LLMs lack domain context, and high-quality demonstrations are expensive to curate [3, 7]. LLM-based index advisors such as LLMIA and LLMIdxAdvis treat demonstration construction as a first-class pipeline stage, synthesizing workloads with a stronger model and labeling them with heuristic ensembles before retrieving the most similar examples at inference time [7, 9].

Retrieval-augmented generation (RAG) addresses the complementary limitation: pretrained parametric knowledge is static, whereas DBA work depends on engine-specific manuals, internal runbooks, historical incidents, and organization-specific conventions [7]. RAG embeds such external documents, retrieves the chunks most relevant to the current query by vector similarity, and injects them into the prompt at inference time. D-Bot builds two knowledge stores — a document store for diagnostic manuals and a feedback store for refinement patterns extracted from past interactions — and retrieves from both during anomaly analysis [3]. DB-GPT generalizes the same pattern into a bilingual RAG pipeline with pluggable embedding and reranking components [1]. RAG also provides a mechanism for grounding LLM output against verifiable sources, which partially mitigates hallucination and which matters when recommendations are destined for production systems.

Tool use closes the loop between the LLM and the database. Rather than asking the model to reason about cost or cardinality from first principles, the system exposes database primitives — EXPLAIN, the what-if caller, query execution, metric collection — as callable tools and lets the LLM invoke them as part of its reasoning. This is the mechanism by which an LLM-based advisor can escape the pure-reasoning regime that zero-shot baselines occupy: LLMIdxAdvis invokes the what-if caller to self-optimize its recommendations across iterations [9], λ-Tune runs candidate configurations against the target DBMS through a configuration evaluator that timeouts inefficient choices [5], and D-Bot's diagnosis experts call index optimizers and monitor metrics as part of their reasoning chains [1]. Tool use is also what makes the distinction between LLMs and classical advisors partly a false dichotomy in practice: LLM-driven systems frequently retain what-if estimation as an inner loop and differentiate themselves on how the outer loop reasons over its results.

Multi-agent orchestration decomposes workflows that are too complex for a single prompt into interacting specialized roles. Recent work has shown that a single zero-shot LLM call underperforms on index recommendation even relative to heuristic baselines, GPT-4o trails Extend by 15.8% on average in one controlled comparison [3], and that decomposing the task into planning, selection, combination, and revision sub-steps recovers and surpasses that performance. MAAdvisor implements this pattern explicitly for index selection, with a planning agent and a reflection agent coordinating low-level selection, combination, and revision agents under a storage budget [7]. D-Bot applies the same pattern to diagnosis, assigning domain experts (CPU, I/O, workload, configuration) whose outputs are cross-reviewed before report generation [7]. DB-GPT orchestrates data-analyst, engineer, and architect roles through a standard-operating-procedure coordinator [9]. The common motivation across these systems is that complex DBA problems benefit from role specialization and explicit verification steps, both of which are difficult to encode in a single prompt.

III. LANDSCAPE OF LLM-DRIVEN DBA SYSTEMS

The systems surveyed in this paper occupy a space that is easier to navigate with a small set of axes than with a linear narrative. Four dimensions recur across the literature and together locate each system relative to its peers: the task scope it addresses, the LLM integration paradigm through which it uses the model, the deployment model under which it runs, and the autonomy level at which it interacts with the human user. Table 1 maps the ten core systems onto these axes, and the remainder of this section explains why these axes were chosen and what the distribution across them reveals.

Task scope ranges from narrow to holistic. At one end, systems such as LLMIA, LLMIdxAdvis, and MAAdvisor target a single physical-design task, index recommendation, and are therefore directly comparable to classical advisors such as DTA and AutoAdmin. At the other end, D-Bot and DB-GPT position themselves as copilots spanning diagnosis, knowledge retrieval, natural-language interfaces, and tool invocation. λ-Tune sits in between by producing whole configuration scripts that combine index recommendations with system-knob settings, and the schema-oriented systems (LLMDap, the schema-linking analysis, and the LLM-based database design tool) target data-modeling and metadata tasks that classical automation has historically left to humans. The distribution is informative on its own: narrow single-task systems cluster around the two anchor tasks of this survey, while broader copilots remain comparatively rare.

LLM integration paradigm captures how the model is conditioned on the task. Five patterns appear in the literature: zero-shot prompting, in-context learning (ICL) with curated demonstrations, fine-tuning on domain-specific corpora, agentic use with tool calls and reasoning traces, and multi-agent orchestration with specialized roles. These are not mutually exclusive. D-Bot layers agentic reasoning over a RAG-indexed knowledge base and coordinates multiple experts; DB-GPT combines fine-tuned text-to-SQL with RAG and a multi-agent layer; LLMIdxAdvis uses ICL with demonstration retrieval plus iterative tool use through a what-if caller. The distinction that matters for this survey is whether a system treats the LLM as a single reasoner or decomposes the task across coordinating agents, because that choice correlates with both reliability and cost.

Deployment model separates systems that assume cloud-hosted LLMs accessed over a public API from those designed to run on private or on-premise infrastructure. Cloud deployment simplifies access to frontier models but raises data-governance concerns when production database contents appear in prompts. DB-GPT is the clearest example of a private-deployment design, emphasizing fine-tuned open models that can run on local servers without internet connectivity. Most index advisors surveyed here are paradigm-agnostic in principle but are evaluated against cloud models such as GPT-4, GPT-4o, or

GPT-4o-mini, which leaves open-weights deployment as an unresolved question for production use.

Autonomy level describes where the human sits in the loop. Advisory systems produce recommendations that a DBA reviews before applying, conversational systems interact through dialogue and incorporate user feedback into refinement, and fully automated systems apply changes without human intervention. Every system surveyed here operates at advisory or conversational autonomy. None claims full automation for schema or index changes in production, which is itself a finding: the field has converged on recommendation generation with human verification as the operating point, reflecting the validation and trust limitations discussed in Section 7.2.

The axes are not independent. Narrow-scope advisors tend to use ICL with tool feedback and operate at advisory autonomy; holistic copilots tend to combine RAG with multi-agent orchestration and operate at conversational autonomy; schema-oriented systems are dominated by prompting and RAG with limited tool use. Where a system sits on one axis predicts a lot about where it sits on the others, which is one reason the taxonomy is short rather than a full cross-product.

TABLE I
TAXONOMY OF SURVEYED LLM-DRIVEN DBA SYSTEMS

System	Task.Scope	LLM.Paradigm	Deployment	Autonomy
D-Bot	Diagnosis / copilot	RAG + multi-agent	Cloud / Hybrid	Advisory
DB-GPT	NL interface / analytics	Fine-tune + RAG	Cloud / Hybrid	Advisory
λ-Tune	System tuning	Prompting	Cloud / Hybrid	Advisory
LLMIA	Index recommendation	ICL feedback	Cloud / Hybrid	Advisory
LLMIdxAdvis	Index recommendation	ICL loop	Cloud / Hybrid	Advisory
MAAdvisor	Index recommendation	Multi-agent	Cloud / Hybrid	Advisory
DTA study	Index rec.	Zero-shot	Cloud / Hybrid	Advisory
LLM-CDM tool	Schema design	Zero-shot	Cloud / Hybrid	Advisory
LLMDap	Data profiling	RAG	Cloud / Hybrid	Advisory
Schema-linking	Schema linking	Fine-tune	Cloud / Hybrid	Advisory

IV. LLMs as Holistic DBA Copilots

The systems surveyed in this section share a defining ambition: rather than targeting a single physical-design task, they attempt to automate broader segments of the DBA workflow by treating the LLM as a general-purpose reasoning engine, grounded in domain-specific context and connected to database tools. Three systems illustrate the design space: D-Bot and DB-GPT address diagnosis and multi-task assistance through conversational interaction, while λ-Tune targets system-wide configuration tuning. Comparing them reveals a shared architectural pattern (prompt-grounded reasoning over retrieved context plus tool invocation) alongside sharp differences in scope, grounding mechanism, and verification strategy.

A. Diagnosis and end-to-end assistance

D-Bot is the most explicitly diagnosis-oriented system in the survey. Given an anomaly detected by metric monitoring, it follows a four-stage pipeline: expert assignment, expert diagnosis, group discussion, and report generation [1]. In the first stage, an LLM analyzes the anomaly description and selects relevant domain experts (e.g., a CPU expert, a workload expert). Each assigned expert then reasons independently using multi-step tool calls and knowledge retrieval, guided by a tree search algorithm that explores alternative reasoning paths and selects among them through LLM-based reflection. After individual diagnosis, experts exchange intermediate results in a cross-review step, and a final LLM call synthesizes a report specifying root causes, solutions, and references to source documents. Two knowledge stores ground the process: a document store built by extracting knowledge chunks from diagnostic manuals, and a feedback store that accumulates "if-then" refinement patterns from past user interactions [3].

D-Bot's self-refinement mechanism is notable. When a user provides feedback (e.g., "offer more in-depth analysis"), the system extracts the underlying intent, generates executable test suites (programmatic checks such as return ask_llm("Does it provide in-depth analysis of causes?")), and iteratively refines the LLM response until all tests pass or a retry threshold is reached [7]. The refined patterns are then stored for future reuse, making D-Bot a self-evolving system in practice. On 51 real single-cause anomalies, D-Bot powered by GPT-4 achieves an F2-score of 67.6% in fully automatic mode. On 9 multi-cause anomalies, it reaches 59.5% without user intervention, a score the authors report as exceeding that of human DBAs on the same set [9].

DB-GPT occupies a wider but shallower niche. Where D-Bot is deep on diagnosis, DB-GPT spans natural-language query interfaces, multi-source knowledge-base question answering, text-to-SQL generation, and generative data analytics [1]. Its architecture is organized around three pillars. First, a multi-source RAG pipeline ingests documents, web pages, and database content, then retrieves and reranks relevant chunks for adaptive in-context learning. Second, a Service-oriented Multi-model Framework (SMMF) supports deploying and serving multiple LLMs on varying inference backends (vLLM, HuggingFace Transformers, TGI, TensorRT), enabling model swaps without changing application logic [7]. Third, a multi-agent layer assigns specialized roles (data analyst, software engineer, database architect) coordinated through standard operating procedures inspired by MetaGPT [10].

The sharpest contrast between the two systems lies in deployment philosophy. D-Bot assumes cloud-hosted frontier models (GPT-4 by default) and focuses its engineering on the reasoning pipeline and the knowledge-grounding loop. DB-GPT is designed explicitly for private deployment: it can run on local servers without internet connectivity, applies proxy de-identification to mask personal identifiers in data processing, and fine-tunes open models on domain corpora [3]. This makes DB-GPT the only surveyed copilot to address the data-governance constraint that blocks many enterprises from sending production database contents to third-party

APIs, a theme taken up further in Section 7.3.

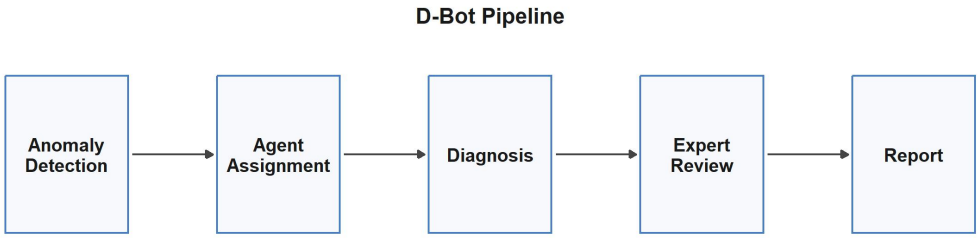

Fig. 1 D-BOT ARCHITECTURE DIAGRAM

B. *System tuning*

λ-Tune differs from both systems above in a way that matters structurally for this survey: it treats the LLM not as a conversational agent but as a one-shot configuration generator followed by a principled evaluation engine [5]. The input is an OLAP workload, a hardware specification (cores and memory), and the name of the target DBMS. λ-Tune compresses the workload into a representation that focuses on join structure rather than full SQL text, embeds this compressed representation into a prompt alongside hardware details, and invokes the LLM k times at varying temperatures to obtain k candidate configurations [9]. Each configuration is a complete script of ALTER SYSTEM SET and CREATE INDEX commands, not a set of individual knob hints, which is how λ-Tune differs most sharply from prior learned tuners such as DB-BERT and GPTuner that extract single-parameter suggestions and navigate the combinatorial space through trial runs [7].

Two engineering contributions distinguish λ-Tune from a naive baseline of prompting and then running the best output. First, the workload compressor treats prompt generation as a cost-based optimization problem under a token budget, selecting the most informative join-structure snippets while bounding API fees. Ablation studies show that the compressed representation achieves better performance than sending full SQL queries, even with a ten-fold token reduction [1]. An obfuscation experiment, in which table and column names were replaced with generic identifiers, provides evidence that λ-Tune is not simply memorizing configurations from pre-training data [5]. Second, the configuration evaluator uses a multi-round timeout scheme with provable guarantees: total evaluation time is bounded by $O(k \cdot \alpha \cdot C\_best)$, where C_best is the execution time of the best configuration and α is the timeout growth factor [7]. Indexes are created lazily and ordered by a dynamic-programming scheduler to minimize reconfiguration overhead [9].

λ-Tune evaluates on PostgreSQL and MySQL using TPC-H, JOB, and TPC-DS, and demonstrates consistent advantage over UDO, GPTuner, DB-Bert, LlamaTune, and ParamTree, while requiring only 5 configuration evaluations compared to hundreds for most baselines [10]. When restricted to index recommendation alone, however, λ-Tune generally does not match specialized index advisors such as Dexter and the DB2 Index Advisor, with the exception of TPC-DS. This result is expected given λ-Tune's broader scope, and it also sets up the motivation for the dedicated index recommendation systems discussed in Section 5.

C. *What copilots do well, where they break down*

Taken together, the three copilots reveal a consistent pattern of strengths. All three demonstrate that LLMs can operate across the boundaries of tasks that classical automation keeps siloed: D-Bot combines knowledge retrieval, tool invocation, multi-expert reasoning, and report generation in a unified

pipeline; DB-GPT spans query interfaces, analytics, and operational support; λ-Tune generates configurations that blend knob tuning with physical design in a single prompt response. The ability to work in natural language, consume unstructured documentation, and generate human-readable reports constitutes a qualitative advance over classical tools that take structured inputs and produce structured outputs without explanation.

The failure modes are equally consistent. First, all three depend on the LLM backbone in ways that create reliability concerns. D-Bot reports that its LLM tends to assign all possible experts even for simple anomalies, wasting resources, and that experts sometimes repeat each other's reviews during group discussion unless corrected by user feedback [13]. λ-Tune requires multiple LLM invocations precisely because individual outputs vary with temperature and cannot be assumed correct. Second, none of these systems operates at fully automated autonomy. D-Bot is explicitly conversational and expects user feedback for refinement; λ-Tune generates candidates that must be validated by actual execution; DB-GPT positions itself as a copilot rather than an autonomous agent. Third, cost and latency remain open issues. D-Bot's multi-expert pipeline with tree search reasoning multiplies LLM calls relative to simpler architectures. λ-Tune's evaluation can be bounded formally but still involves materializing configurations and running full workloads. DB-GPT partially mitigates API cost by deploying local models but trades frontier-model accuracy for deployment flexibility. These trade-offs define the operating envelope of copilot-class systems and motivate the narrower, more verifiable approaches covered in Sections 5 and 6.

V. LLMs for Index Recommendation

Index recommendation is the task where LLM-based approaches have produced the densest body of work and where the comparison with classical automation is sharpest. This section traces the design space from problem formulation through three distinct LLM integration strategies, then confronts them with an independent empirical evaluation against a production-grade baseline. The section closes with a cross-system comparison that identifies the trade-offs any practitioner deploying these systems would face.

A. Problem setting and classical baselines

The index selection problem (ISP) asks, given a workload W of m queries, a set of index candidates I derived from columns appearing in the workload, and a storage constraint S_c, for the subset I* of I that minimizes total workload cost while respecting the budget [7]. The problem is NP-complete [15], so all practical solvers approximate it through some combination of candidate generation, index selection, and benefit evaluation.

Classical solvers split into two families. Heuristic advisors (AutoAdmin, DTA, DB2Advis, Extend, Relaxation, Drop) apply greedy, incremental, or relaxation-based search, exploring the configuration space through thousands of benefit evaluations. On TPC-DS, heuristics such as AutoAdmin, DTA, Relaxation, and Drop require over 20,000 benefit evaluations in a single tuning run [7]. Learning-based advisors (AutoIndex, DQN, SWIRL, BALANCE, MFIX) reformulate index selection as a sequential decision task using Monte Carlo tree search, Deep Q-Networks, proximal policy optimization, or Bayesian optimization, typically requiring 100 to 400 iterations [17]. Both families depend on the what-if caller, an interface that materializes hypothetical indexes virtually and returns EXPLAIN-estimated costs without building them on disk [19].

The core limitation motivating LLM-based alternatives is the disconnect between estimated cost and actual query latency. Configurations that minimize what-if estimated cost do not always minimize measured execution time, and heuristic advisors have been observed to produce performance regressions on specific query templates even when overall estimated-cost improvement is positive [1, 9]. This gap

between estimation and reality is the structural opening that the LLM-based systems surveyed below attempt to exploit, each through a different mechanism.

B. *In-context and resource-efficient advisors*

LLMIA and LLMIdxAdvis share a two-stage architecture: an offline demonstration-construction pipeline followed by an online ICL-driven recommendation pipeline. Despite this common skeleton, they differ in how they formulate the LLM's role and how they close the feedback loop with the database.

Both systems construct demonstrations offline by synthesizing diverse OLAP workloads using GPT-4-Turbo (approximately 1,000 SQL queries per database schema) and labeling them with an ensemble of heuristic methods to generate optimal index sets under varying storage constraints [13, 9]. Each demonstration is a quadruple of database context, workload, current indexes, and refinement action. To handle different starting states, both systems distinguish between initial refinement demonstrations (starting from no indexes) and incremental refinement demonstrations (starting from existing index configurations). At inference time, the most relevant demonstrations are retrieved by cosine similarity over workload meta-features and injected into the prompt alongside a system instruction and detailed input information [,].

Where the systems diverge is in their inference-time strategies. LLMIA operates in two modes: Cost-Instruct (LLMIA(C)), which uses the what-if caller and estimated cost as feedback, and Latency-Instruct (LLMIA(L)), which replaces virtual index management with actual index builds and measured workload latency as the feedback signal [7]. LLMIA(L) requires fewer DBMS interactions than classical advisors, making actual-latency feedback practical for important or recurring workloads. Across five OLAP benchmarks (TPC-H, JOB, TPC-DS, SSB, and two real-world workloads) under varying storage constraints, LLMIA(L) consistently achieves satisfactory or near-optimal actual latency reduction compared to all baselines 1[7]. The system also demonstrates robustness across LLM backbones: replacing GPT-4o-mini with models including GPT-4.1-mini, DeepSeek-V3, LLaMA-3.3-70B, and several Qwen variants still yields promising results [3]. Stability tests over 10 trials show a maximum latency reduction of 22.14% and a minimum of 16.55%, both exceeding most baselines [9].

LLMIdxAdvis focuses on resource efficiency and introduces two inference-scaling dimensions that distinguish it from LLMIA. Vertical scaling combines multi-sampling (temperature 0.6, 8 samples) with an "Index-Guided Major Voting" strategy: candidate indexes from multiple LLM outputs are aggregated by recommendation frequency, and a potential option is constructed by retaining high-frequency CREATE and DROP statements [11]. This addresses the stochastic variance of LLM outputs. Horizontal scaling implements a self-optimization loop: after each recommendation round, the prompt is updated with the current index state and database feedback (cost fluctuation, used indexes from query plans), allowing the LLM to iteratively refine its own output [7]. LLMIdxAdvis positions itself as the first LLM-based workload-level index advisor, exploiting long-context capabilities to reason about inter-query index relationships rather than treating each query in isolation [7].

A key finding shared by both systems is that zero-shot prompting without demonstrations yields substantially degraded performance. LLMIdxAdvis's ablation studies show progressive decline as the number of demonstrations decreases, confirming that general-purpose LLMs lack sufficient database-specific knowledge to recommend indexes without curated examples [7]. This offline preparation cost, while amortized as a one-time investment, is nontrivial: for LLMIA, the total API cost for demonstration construction across three schemas is roughly $90 ($49 input, $41 output), and label generation takes approximately one day when distributed across 10 CPU servers [19].

C. *Multi-agent and zero-shot approaches*

MAAdvisor (referred to as AMAZe in its

paper) takes a fundamentally different stance: it operates in a zero-shot setting, requiring no offline demonstration construction and no database-specific training [1]. The motivation is explicit: preliminary experiments show that a single zero-shot GPT-4o call underperforms the heuristic baseline Extend by an average of 15.8%, confirming that index recommendation is too complex for a single LLM invocation [3]. AMAZe addresses this by decomposing the problem into a hierarchical multi-agent pipeline with five specialized agents.

The pipeline begins with a workload representation step that transforms the raw workload into a compact set of column candidates. Rather than encoding full queries (the TPC-DS workload with 83 queries exceeds 36,000 tokens), AMAZe aggregates all index-relevant information at the column level, including operator types, estimated cardinality, column distribution, estimated utility, and storage cost [9]. This representation bounds input length by the number of database columns rather than by query count, making it independent of workload complexity.

At the high level, a planning agent decomposes the recommendation into sub-steps, choosing at each iteration whether to invoke the selection agent (add a single-column index), the combination agent (merge single-column indexes into composite indexes), or the revision agent (remove hallucinated, redundant, or regressive indexes) [11]. A reflection agent evaluates the planning path after each step and provides corrective suggestions to prevent the planner from looping or hallucinating invalid actions [13]. The combination agent is grounded in external knowledge (index combination rules, expert experience) and also serves to mitigate optimizer estimation bias by transforming individually suboptimal indexes into composite indexes that better capture multi-column access patterns [13]. The revision agent applies a three-fold justification mechanism drawing on column information, expert experience, and a trained regression indicator that predicts whether a given index set would cause workload-level performance degradation [15].

Experimentally, AMAZe achieves the highest benefit-to-cost ratio in 10 of 16 settings across TPC-H, TPC-DS, DSB, and JOB under four storage budgets, and remains in the top tier in the remaining settings [19]. Ablation results on TPC-H are sharp: removing the multi-agent pipeline (one-call recommendation) drops performance from 61.33% average relative workload improvement to 11.20%; removing the workload representation drops it to 29.52%; removing both drops it to 10.58% [7]. The system generalizes across LLM backbones: GPT-3.5-turbo (60.01%) and QWen3-32b (60.34%) match GPT-4o performance, while smaller 14B models achieve roughly 40% [9]. Notably, DeepCoder-14b, a code-specialized model, underperforms QWen3-14b at the same parameter size, suggesting that reasoning ability matters more than coding ability for this task [1].

D. Empirical reality check against DTA

The most rigorous head-to-head evaluation of LLM-based index tuning against a production baseline comes from a study comparing GPT-5 with Microsoft's Database Tuning Advisor (DTA) on four real customer workloads and TPC-H (sf=10) on Microsoft SQL Server [3]. This study differs from the academic evaluations above in three ways: it uses real-world analytical queries with CTEs, views, and numerous pre-existing expert-created indexes; it evaluates on a commercial DBMS rather than PostgreSQL; and it adopts actual execution time rather than estimated cost as the primary metric.

For single-query workloads, GPT-5 performs comparably to or better than DTA for most queries across all five workloads. On Real-D, the LLM significantly outperforms DTA on many queries, while on Real-S, performance is roughly equivalent. Anonymizing table and column names does not degrade GPT-5's performance, consistent with earlier findings that LLM performance does not depend on benchmark memorization [5]. DTA itself produces query performance regressions (QPRs) in several cases (e.g., a nearly 10x slowdown on Real-R query 22), where the

LLM identifies better plans [7].

For multi-query workloads, the picture changes. DTA provides more stable and reliable improvements overall, while GPT-5's performance varies considerably across workloads and invocations [1]. On Real-D with K=10, all five LLM invocations substantially outperform DTA, with the best achieving 4x further speedup. But on Real-M, GPT-5 consistently fails to identify the two bottleneck queries (accounting for 85% of execution time), and on Real-S, most LLM responses yield little improvement [3]. A key insight is that LLM recommendations tend to benefit multiple queries simultaneously rather than targeting single bottlenecks, which produces high-quality candidate indexes that DTA misses but also leads to omissions when the workload is dominated by a few expensive queries [5].

Two integration experiments reveal the practical difficulty of combining the approaches. Enriching DTA's candidate pool with LLM-recommended indexes often leads to performance degradation, because DTA's cost-based selection amplifies estimation errors on the expanded candidate set [7]. Validation-based integration (materializing configurations and measuring actual execution time) is more robust but reveals a severe cost bottleneck: the overhead of building recommended indexes and executing the across benchmarks due to its fixed demonstration count, while AMAZe's cost grows with schema complexity (4x more tokens on TPC-DS than TPC-H) [2]. API costs remain moderate for all LLM-based systems (under $0.20 per recommendation for AMAZe on complex benchmarks) but become relevant at scale.

On robustness, the evidence is mixed. LLMIA(L) with actual-latency feedback produces the most reliable results by directly optimizing for the metric that matters, bypassing cost-model inaccuracy entirely. AMAZe's revision agent with its regression indicator mitigates the index-regression problem that afflicts both heuristic and prompt-based methods: it avoids the severe workload for validation is often significantly higher than the tuning cost itself, and the diverse recommendations that LLMs produce amplify this overhead [9].

E. Cross-system comparison and trade-offs

The LLM-based index advisors surveyed in this section occupy distinct positions in a space defined by three axes: grounding strategy, efficiency, and robustness.

On grounding, LLMIA and LLMIdxAdvis invest in offline demonstration construction to inject database expertise through ICL, then close the loop with what-if or actual-latency feedback. AMAZe replaces demonstrations with a structured workload representation and multi-agent decomposition, operating zero-shot. The DTA evaluation study uses raw prompting with schema and plan context and no iterative refinement. The trade-off is between offline preparation cost (one-time but nontrivial) and inference-time complexity (multi-agent coordination with more LLM calls per recommendation).

On efficiency, AMAZe presents a clear advantage for complex workloads: its input length scales with column count rather than query count, while heuristic baselines like Relaxation can take over 100x longer on TPC-DS (83 queries) than on TPC-H (19 queries) [1]. LLMIdxAdvis maintains stable input lengths

regressions observed with DB2Advis (nearly 50% regression on TPC-H query 9) and the cross-benchmark performance fluctuation seen with LLMIdxAdvis [3, 5]. The DTA study, however, documents that even GPT-5 exhibits high variance across invocations on multi-query workloads and fails on workloads dominated by a few expensive queries [7]. No LLM-based system yet matches DTA's consistency on real customer workloads, though several surpass its best-case performance.

The gap that no system fully closes is the tension between recommendation quality and validation cost. LLMIA(L)'s actual-latency mode addresses cost-model inaccuracy but adds the overhead of physical index creation

and workload execution. The DTA evaluation study quantifies this overhead as often exceeding the tuning cost itself [9]. Any production deployment of LLM-based index recommendation will need to resolve this validation bottleneck, either through cheaper validation mechanisms, tighter coupling with learned cost models, or a hybrid architecture that uses LLM recommendations to expand the candidate pool while relying on improved cost estimation for selection

VI. LLMs for Schema Design and Understanding

Schema design sits upstream of the physical-design tasks surveyed in Section 5. Where index recommendation operates on an existing schema and workload, the tasks covered here operate on requirements, documents, or natural-language queries and produce or refine the schema itself. Three lines of work apply LLMs to this space: requirements-to-schema generation, data profiling and metadata enrichment, and schema linking for text-to-SQL. The common thread is that all three treat the LLM as a translator between unstructured or semi-structured inputs and the structured representations that databases require.

A. Requirements-to-schema generation

Conceptual database modeling (CDM) translates textual domain requirements into entity-relationship or class-diagram representations. Unlike the subsequent transformation steps (logical and physical design), which are largely mechanical, conceptual design is inherently iterative and has resisted full automation since the idea was first proposed in the early 1980s [13]. Existing approaches rely on NLP-based heuristic rules to extract entities and relationships from natural-language text, but they are typically limited to a single source language and do not produce complete, correct schemas from arbitrary specifications [15].

The LLM-based CDM generator of Divljan and Brdjanin [11] is, to the authors' knowledge, the first online web-based tool that performs automatic CDM derivation. The tool applies a zero-shot prompting approach: the user provides a textual specification and selects one of three LLMs (Gemini-2.5-Flash, Llama-3-70b, Mistral-Large-2, chosen after a preliminary assessment of 16 models), and the system constructs a carefully tailored prompt that instructs the LLM to generate a PlantUML class diagram strictly from the information in the specification, without introducing prior knowledge or assumptions [17]. Post-processing corrects syntactic issues in the generated script, which is then rendered via an external PlantUML service. The prompt enforces structural constraints: every class must contain at least one attribute, inheritance cannot be represented via associations, and enumerations are permitted only when all values are explicitly listed in the source text [19].

The tool's key distinguishing feature is multilingual support: it derives CDMs from specifications in languages with complex morphology (the paper demonstrates Serbian alongside English) without requiring language-specific rules [1]. Two limitations constrain its current scope. First, only the conceptual design step is implemented; forward engineering from CDM to a physical database schema is planned but not yet realized. Second, the evaluation is preliminary, based on illustrative examples rather than systematic benchmarks, so the tool's accuracy on complex or ambiguous specifications remains an open question.

From a survey perspective, requirements-to-schema generation is the least mature of the three schema-related tasks. The state of the art consists of early prototypes rather than evaluated systems, and no study yet benchmarks LLM-generated schemas against human-designed references on a shared dataset. This stands in contrast to the index recommendation literature, where multiple systems are evaluated on the same benchmarks under comparable conditions.

B. Data profiling and metadata enrichment

LLMDap addresses a different point in the schema lifecycle: given an existing dataset and a target metadata schema, it generates structured, interoperable metadata profiles by extracting information from associated documents such as scientific papers and lab protocols [5]. The motivation is the chronic underproduction of metadata in data-sharing ecosystems, where keyword-based search fails because published dataset descriptions do not match the terms used in discovery queries.

The LLMDap pipeline consists of five components [7]. First, auxiliary data referencing identifies the expected value range for each metadata field, drawing on predefined tags, domain-specific ontologies, or example values from the input schema. Second, document chunking and embedding splits the source document by section structure and embeds chunks into a shared vector space. Third, context retrieval performs semantic matching between chunk embeddings and auxiliary-data embeddings to identify the most relevant passages for each schema field. Fourth, LLM-based query constructs a prompt from the retrieved context and the field description, and the LLM generates a prediction for that field. Fifth, pipeline configuration allows adaptation to different domains by swapping LLMs, embedding models, or schema standards [9].

The system is deliberately LLM-agnostic and domain-independent, though the published validation targets the biomedical domain using Beacon v2 schemas and PubMed/Europe PMC corpora [11]. The generated profiles are designed for sharing via data catalogues and marketplaces, and the pipeline supports not only profiling but also question answering and summarization over catalogued datasets through a consumer-facing chat interface [13].

LLMDap's position in this survey is distinctive: it is the only system that applies LLMs to metadata generation rather than to query processing or physical design. Its RAG-based architecture shares the retrieval-then-generate pattern with D-Bot and DB-GPT (Section 4), but its grounding sources are scientific documents and ontologies rather than diagnostic manuals or database logs. The system's practical limitation is the absence of quantitative accuracy benchmarks against manually curated profiles, which makes it difficult to compare directly with the evaluated systems in other sections of this survey.

C. Schema linking and its role in physical design

Schema linking, the task of identifying which database tables and columns are relevant to a given natural-language question, is a critical component of text-to-SQL pipelines [15]. Though it is not a traditional DBA task, schema linking has direct implications for physical design: a schema linking model that consistently identifies which columns participate in query predicates, joins, and groupings is, in effect, performing the same column-relevance analysis that index advisors require. This connection motivates its inclusion in a DBA-oriented survey.

The most comprehensive analysis of LLM-based schema linking comes from Katsogiannis-Meimarakis et al. [17], who evaluate multiple LLMs under four usage scenarios on the Spider and BIRD benchmarks: few-shot prompting, few-shot prompting with question decomposition, supervised fine-tuning, and fine-tuning with question decomposition. The study yields several findings with implications for physical design.

First, on precision and recall trade-offs: question decomposition consistently increases recall (sometimes by over 20%) while decreasing precision, and supervised fine-tuning increases precision while decreasing recall [19]. For physical design, recall is the critical dimension, because a missed column means the corresponding index or join optimization cannot be considered. Second, model size is not a reliable predictor of quality: a small code-oriented model like DeepSeek-Coder-6.7B is competitive with and sometimes superior to larger alternatives, particularly after fine-tuning [1]. This has cost implications for any system that embeds schema linking as an inner component. Third, the study introduces a schema enrichment investigation that provides additional columns beyond the

strict minimum needed for SQL generation, using diversity-based or representativeness-based sampling strategies [3]. Enriched schemas can improve downstream text-to-SQL accuracy, with Deepseek-33B gaining over 5 percentage points on BIRD when enriched oracle links are provided [5]. The finding that the traditional "oracle" schema (containing only the minimal necessary columns) is not always optimal for SQL generation challenges a longstanding assumption in the text-to-SQL literature [7].

Fourth, prediction refinement techniques that inject database structural knowledge (foreign-key connections, parent-table resolution, fuzzy matching) boost average recall on BIRD by 26.33% at a cost of only 4.75% in precision [9]. This result demonstrates that LLMs' generative strength can be complemented by rule-based post-processing that encodes relational constraints the model tends to overlook.

For the broader DBA automation agenda, schema linking represents a bridge between the natural-language interface layer and the physical-design layer. A system that accurately links queries to schema elements could, in principle, feed column-usage statistics into an index advisor, transforming schema linking from a text-to-SQL preprocessing step into a workload-analysis component. No surveyed system yet implements this pipeline end to end, leaving it as an open direction discussed further in Section 8.

VII. CROSS-CUTTING THEMES

Sections 4 through 6 examined individual systems organized by task. This section pulls threads that run across those boundaries: the recurring architectural patterns through which LLMs are integrated into DBA workflows, the failure modes that arise when LLM outputs feed into production systems, and the deployment constraints that determine whether a given system can be used in practice.

A. Prompting, RAG, and tool integration patterns

Three integration patterns recur across the surveyed systems, and every system combines at least two of them.

Prompt engineering with structured context injection. All surveyed systems embed database-specific information into the prompt, but the strategies for selecting and compressing that information vary substantially. λ-Tune treats prompt generation as a cost-based optimization problem, compressing the input workload into join-structure snippets and selecting the most informative ones under a token budget [11]. LLMIdxAdvis and LLMIA inject workload features (WHERE predicates, JOIN columns, GROUP BY/ORDER BY columns, number of distinct values) alongside dynamically matched demonstrations [13, 15]. AMAZe takes the most aggressive compression approach, reorganizing the entire workload at the column level so that input length scales with column count rather than query count [17]. The DTA evaluation study takes the opposite stance, deliberately retaining complete query information to avoid eliminating critical details, relying on GPT-5's million-token context window to accommodate the full workload [19]. The CDM generator constrains the LLM through a carefully designed zero-shot prompt that enforces structural rules for the output schema (e.g., prohibiting empty classes, requiring explicit primary keys) [1]. Across these systems, prompt design is not an afterthought but an engineering component whose quality directly shapes recommendation quality.

Retrieval-augmented generation. RAG appears wherever the system needs access to knowledge that is absent from the LLM's parametric weights. D-Bot maintains two knowledge stores (document store and feedback store) and retrieves relevant chunks by embedding similarity during diagnosis [11]. DB-GPT generalizes RAG into a multi-source pipeline ingesting databases, web pages, and PDFs, with pluggable embedding and reranking strategies [13]. LLMDap applies the same pattern to metadata generation, retrieving relevant document chunks matched against ontology-guided auxiliary data for each schema field [15]. In all three cases, RAG

serves a dual purpose: injecting domain knowledge and providing a verifiable grounding source that partially mitigates hallucination. The feedback store in D-Bot extends this pattern into a learning loop, where refinement patterns extracted from past user interactions are re-injected into future prompts [17].

Tool use and database feedback loops. The distinction between advisory and grounded systems often reduces to whether the LLM can call database primitives and incorporate their results. LLMIdxAdvis invokes the what-if caller to estimate benefits and updates the prompt with cost-fluctuation feedback across iterations [1]. LLMIA offers two modes: one using what-if estimated cost, another using actual workload latency as feedback, the latter bypassing cost-model inaccuracy at the expense of physical index creation [3]. λ-Tune runs candidate configurations against the actual DBMS through a multi-round evaluator with provable time guarantees [9]. D-Bot's diagnosis experts call index optimizers and monitor metrics as part of their reasoning chains [11]. The pattern that emerges is that tool use converts the LLM from a one-shot predictor into an iterative optimizer, and the most effective systems are those that close the feedback loop tightest.

B. Hallucination, validation, and safety in production databases

LLM hallucination takes specific forms in the DBA context that differ from the generic text-generation case. An index advisor may recommend indexes on columns that do not exist in the schema, produce redundant recommendations, or suggest configurations that cause performance regression rather than improvement. AMAZe explicitly designs its revision agent to address all three failure modes: a hallucination check identifies non-existent index selections, and a three-fold justification mechanism drawing on column information, expert experience, and a trained regression indicator removes inefficient indexes [13]. AMAZe's structured workload representation further reduces hallucination by reframing index recommendation as a selection task over predefined column candidates, aligning the problem with LLMs' decision-making strengths rather than requiring free-form generation [13].

The DTA evaluation study provides the most detailed account of variance as a safety concern. Across five independent invocations of GPT-5 on the same workload, recommendations differ substantially, and the gap between the best and worst invocations can span from significant speedup to negligible improvement [15]. On multi-query workloads, DTA provides more stable results than GPT-5, whose performance varies considerably across both workloads and invocations [17]. This variance means that even when the best LLM invocation outperforms the classical baseline, the expected-case performance may not, which is a practical barrier to deployment.

Validation addresses the gap between recommendation and verified outcome. Three strategies appear in the literature. First, λ-Tune's configuration evaluator materializes candidate configurations and runs the actual workload, with timeouts that provide provable bounds on total evaluation time [19]. Second, LLMIA(L)'s latency-instruct mode replaces cost estimation with actual execution feedback, trading thoroughness for accuracy [3]. Third, the DTA evaluation study explores enriching DTA's candidate pool with LLM recommendations, but finds that cost-based selection on the expanded pool often degrades performance due to amplified estimation errors [3]. The study's Key Finding 11 quantifies the fundamental tension: the cost of performance validation (index creation plus workload execution) is often significantly higher than the cost of tuning itself, and the diverse recommendations that LLMs produce amplify this overhead [3]. No system surveyed here has resolved this tension in a way that would be acceptable for unattended production deployment.

C. Privacy and on-premise deployment

When an LLM-based DBA system includes production data in its prompts (schema details, query text, execution plans, metric values), that data flows to whatever infrastructure hosts

the model. For cloud-hosted frontier models, this creates a governance conflict: the data may contain proprietary schema structures, business-sensitive query patterns, or personally identifiable information embedded in predicates or join conditions.

DB-GPT is the only surveyed system that directly addresses this constraint through its architecture. It supports deployment on personal devices or local servers without internet connectivity, fine-tunes open models on domain-specific corpora for the text-to-SQL component, and applies proxy de-identification techniques to obscure personal identifiers in data processing modules [3]. Its Service-oriented Multi-model Framework (SMMF) supports multiple inference backends (vLLM, HuggingFace Transformers, TGI, TensorRT), allowing organizations to select an on-premise stack without depending on external APIs [3].

The remaining systems are evaluated against cloud-hosted models (GPT-4, GPT-4o, GPT-4o-mini, GPT-5) and do not include privacy-preserving mechanisms in their architectures. λ-Tune's obfuscation experiment, in which table and column names were replaced with generic identifiers, provides indirect evidence that meaningful recommendations can survive anonymization [3]. AMAZe's workload representation, which aggregates information at the column level and strips raw SQL text, achieves a similar effect by design, though privacy is not its stated motivation [5]. These results suggest that prompt-level obfuscation or abstraction may offer a lightweight path toward privacy-preserving LLM-based DBA tools, but no system has systematically evaluated the privacy-accuracy trade-off.

The gap is clear: the systems with the strongest benchmark results rely on frontier cloud models, while the system with the strongest privacy story (DB-GPT) does not report head-to-head comparisons against the same baselines on the same benchmarks. Whether open-weights models deployed on premise can match the recommendation quality of frontier models in the DBA domain remains an open empirical question, and one with significant practical implications for enterprise adoption.

VIII. Open Challenges and Future Directions

Scaling to large schemas and complex workloads. The systems surveyed here are evaluated predominantly on benchmarks with modest schema sizes: TPC-H has 8 tables, JOB has 21, and even TPC-DS has 25. Real enterprise databases routinely contain hundreds or thousands of tables, and workloads may comprise thousands of heterogeneous queries with CTEs, views, and nested subqueries. AMAZe's column-level workload representation bounds input length by column count rather than query count [7], which helps, but even this representation grows linearly with schema width. The DTA evaluation study's approach of retaining complete query information relies on million-token context windows [9], a strategy that does not yet generalize to open-weights models with shorter limits. Whether multi-agent decomposition, hierarchical summarization, or learned compression can maintain recommendation quality at enterprise scale remains untested.

Tighter coupling with cost models and learned optimizers. A recurring finding across Sections 5 and 7 is that the gap between optimizer-estimated cost and actual query latency limits every system that depends on what-if evaluation. LLMIA's Latency-Instruct mode sidesteps the problem by using actual execution time as feedback [11], but this requires physically building indexes and running the workload, an overhead the DTA evaluation study shows can exceed the tuning cost itself [13]. A promising direction is to integrate learned cost models or learned cardinality estimators into the LLM feedback loop, replacing the what-if caller with a predictor trained on execution traces from the target workload. No surveyed system yet implements this hybrid, and doing so would require bridging two research communities (learned database components and LLM agents) that have developed largely independently.

Workload drift and continual learning. All index advisors surveyed here assume a static workload at recommendation time. Production workloads shift with application releases, seasonal traffic patterns, and schema migrations. AMAZe's ablation on TPC-H 20x (380 queries) shows robust performance under increased workload volume [1], but volume change is not the same as distribution change. D-Bot's feedback store, which accumulates refinement patterns from user interactions [3], offers a lightweight continual-learning mechanism, though its scope is limited to diagnosis. Extending LLM-based advisors to detect drift, trigger re-recommendation, and adapt demonstrations or agent strategies without full offline reconstruction is an open problem.

Trust, explainability, and verifiability of LLM recommendations. None of the surveyed systems operates at fully automated autonomy; all position the human as reviewer or interactor. This is not an accident. LLM-generated index recommendations exhibit high variance across invocations [9], can hallucinate non-existent columns [5], and can cause performance regressions when adopted through cost-based selection [9]. D-Bot addresses part of the trust problem by generating diagnosis reports with references to source documents and knowledge chunks [7], and AMAZe's revision agent provides a structured verification step [7]. But a general framework for certifying that an LLM recommendation will not degrade production performance before it is applied does not yet exist. Developing low-overhead validation techniques that go beyond full workload execution is identified by the DTA evaluation study as a key direction [7].

Standardized benchmarks for LLM-driven DBA tasks. The surveyed systems are evaluated on overlapping but non-identical benchmark sets using different metrics. LLMIA and LLMIdxAdvis report relative workload estimated cost reduction and actual latency reduction on PostgreSQL [,]. AMAZe reports benefit-to-cost ratio on PostgreSQL [7]. The DTA evaluation study uses actual execution time on Microsoft SQL Server [7]. Schema-design and profiling tools lack quantitative benchmarks entirely. This fragmentation makes cross-system comparison difficult and may mask performance differences attributable to the DBMS, hardware, or evaluation protocol rather than to the advisory method. A shared benchmark suite covering multiple DBMS engines, schema sizes, and workload types, with standardized metrics for recommendation quality, efficiency, and safety, would substantially accelerate progress.

IX. CONCLUSION

This survey has examined the emerging body of work applying large language models to tasks traditionally owned by database administrators, with schema design and index recommendation as the two anchoring case studies. We organized the literature along four taxonomy axes (task scope, LLM integration paradigm, deployment model, and autonomy level) and found that the field clusters into three groups: holistic copilots that combine RAG and multi-agent orchestration across diagnosis, tuning, and natural-language interfaces (D-Bot, DB-GPT, λ-Tune); narrow single-task advisors that use in-context learning or multi-agent decomposition for index recommendation (LLMIA, LLMIdxAdvis, AMAZe); and schema-oriented systems that apply prompting or RAG to conceptual modeling, metadata enrichment, and schema linking.

For index recommendation, the surveyed systems demonstrate that LLMs can produce competitive or superior recommendations relative to classical cost-based advisors, particularly when grounded through curated demonstrations, iterative database feedback, or multi-agent decomposition. The evidence is qualified, however: head-to-head evaluation against a production-grade baseline (DTA) reveals high variance across LLM invocations, workload-dependent failures, and a validation-cost bottleneck that currently prevents unattended production deployment. For schema design, the literature is earlier-stage, with prototype tools and domain-specific pipelines but without the shared benchmarks

or systematic evaluation that would permit confident claims about maturity.

Three cross-cutting observations emerge. First, tool use and database feedback loops are what separate effective LLM-based advisors from naive prompting baselines; the tighter the loop, the more reliable the output. Second, hallucination takes domain-specific forms in the DBA context (non-existent indexes, performance-regressing recommendations, schema violations) and requires domain-specific mitigation strategies rather than generic guardrails. Third, the tension between frontier-model capability and on-premise deployment constraints remains unresolved: the strongest results come from cloud-hosted models, but enterprise data governance often demands private infrastructure.

The trajectory of the field points toward hybrid architectures that integrate LLM reasoning with classical cost-model validation, toward shared benchmarks that enable reproducible comparison across systems and DBMS engines, and toward continual-learning mechanisms that adapt to workload drift without full offline reconstruction. Closing these gaps will determine whether LLM-driven database administration moves from a research capability to an operational one.

X. REFERENCES

[1] Zhaoyan Sun, Xuanhe Zhou, Jianming Wu, Wei Zhou, and Guoliang Li. 2025. D-Bot: An LLM-Powered DBA Copilot. In Companion of the 2025 International Conference on Management of Data (SIGMOD/PODS '25). Association for Computing Machinery, New York, NY, USA, 235–238. https://doi.org/10.1145/3722212.3725091

[2] Victor Giannakouris and Immanuel Trummer. 2025. λ-Tune: Harnessing Large Language Models for Automated Database System Tuning. Proc. ACM Manag. Data 3, 1, Article 2 (February 2025), 26 pages. https://doi.org/10.1145/3709652

[3] **LLMIA: An Out-of-the-Box Index Advisor via In-Context Learning with LLMs** Xinxin Zhao, Xinmei Huang, Haoyang Li, Jing Zhang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. "LLMIA: An Out-of-the-Box Index Advisor via In-Context Learning with LLMs." arXiv:2503.07884v2 [cs.DB], March 2025.

[4] **Evaluating the Practical Effectiveness of LLM-Driven Index Tuning with Microsoft Database Tuning Advisor** Xiaoying Wang, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. "Evaluating the Practical Effectiveness of LLM-Driven Index Tuning with Microsoft Database Tuning Advisor." arXiv:2603.09181 [cs.DB], March 10, 2026. Microsoft Research, Redmond, USA.

[5] **LLMIdxAdvis: Resource-Efficient Index Advisor Utilizing Large Language Model** Xinxin Zhao, Haoyang Li, Jing Zhang, Xinmei Huang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. "LLMIdxAdvis: Resource-Efficient Index Advisor Utilizing Large Language Model." *CoRR* abs/2503.07884 (2025).

[6] **LLMIdxAdvis: Resource-Efficient Index Advisor Utilizing Large Language Model** Xinxin Zhao, Haoyang Li, Jing Zhang, Xinmei Huang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. "LLMIdxAdvis: Resource-Efficient Index Advisor Utilizing Large Language Model." *CoRR* abs/2503.07884 (2025).

[7] **Towards an LLM-based Tool for Automated Database Design** P. Divljan and D. Brdjanin. "Towards an LLM-based Tool for Automated Database Design." In *Proceedings of the ER 2025 Posters, Demos, and Workshops (ER25_PAD)*, CEUR Workshop Proceedings, Vol. 4099, 2025. https://ceur-ws.org/Vol-4099/ER25_PAD_Divljan.pdf

[8] **LLMDap: LLM-based Data Profiling and Sharing** Shanshan Jiang et al. "LLMDap: LLM-based Data Profiling and Sharing." In *Proceedings of the VLDB 2025 Workshops*, DEC Workshop, 2025. SINTEF AS. https://www.vldb.org/2025/Workshops/V

LDB-Workshops-2025/DEC/DEC25_5.pdf

[9] **In-depth Analysis of LLM-based Schema Linking** George Katsogiannis-Meimarakis, Katsiaryna Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. "In-depth Analysis of LLM-based Schema Linking." In *Proceedings of the 29th International Conference on Extending Database Technology (EDBT 2026)*, Tampere, Finland, March 24–27, 2026, pp. 117–130. OpenProceedings.org.

[10] **DB-GPT: Empowering Database Interactions with Private Large Language Models** Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, Wang Zhao, Fan Zhou, Danrui Qi, Hong Yi, Shaodong Liu, and Faqiang Chen. "DB-GPT: Empowering Database Interactions with Private Large Language Models." arXiv:2312.17449 [cs.DB], December 2023. https://arxiv.org/abs/2312.17449

[11] Surajit Chaudhuri and Vivek Narasayya. 1998. AutoAdmin “what-if” index analysis utility. In Proceedings of the 1998 ACM SIGMOD international conference on Management of data (SIGMOD '98). Association for Computing Machinery, New York, NY, USA, 367–378. https://doi.org/10.1145/276304.276337

[12] Y. Wu, X. Zhou, Y. Zhang and G. Li, "Automatic Index Tuning: A Survey," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 12, pp. 7657-7676, Dec. 2024, doi: 10.1109/TKDE.2024.3422006.

[13] **Reflexion: Language Agents with Verbal Reinforcement Learning** Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. "Reflexion: Language Agents with Verbal Reinforcement Learning." In *NeurIPS 2023*. arXiv:2303.11366.

[14] **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks** Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In *NeurIPS 2020*, pp. 9459–9474.

[15] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 146–155.

[16] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[17] **Language Models are Few-Shot Learners (GPT-3)** Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, et al. "Language Models are Few-Shot Learners." In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pp. 1877–1901. Curran Associates, 2020. arXiv:2005.14165.

[18] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool that "Reads the Manual". In Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 190–203. https://doi.org/10.1145/3514221.3517843

[19] **LLM As DBA (D-Bot vision paper)** Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. "LLM As DBA." arXiv:2308.05481 [cs.DB], August 2023.