

Decomposing the Task: A Framework for Static and Dynamic Decomposition in Multi-Agent Systems of Small Language Models

Pasquale Miranda^{1,*}, Walter Balzano¹ and Alberto Moccardi¹

¹*Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione (DIETI), Università degli Studi di Napoli "Federico II", Italy*

Abstract

The dominant discourse on agentic artificial intelligence implicitly assumes a centralised design in which a single, large, general-purpose language model handles every task, optionally supported by external tools. This is an architectural choice rather than a necessity. The multi-agent systems (MAS) tradition has long argued that distributed collections of specialised agents can tackle complex problems in structurally different—and often more effective—ways than monolithic solutions. The emergence of capable yet specialised Small Language Models (SLMs) makes this classical position newly relevant in practice. For coordinated SLM ensembles to deliver on their promise, we argue that the central architectural question is not how agents communicate, but how a complex task is *decomposed* into sub-tasks that can be delegated to specialists. This paper contributes a conceptual framework for task decomposition in multi-agent SLM systems. We characterise the problem in ways that distinguish it from both classical MAS decomposition and intra-model prompting strategies; we articulate two regimes, *static* decomposition designed ahead of time and *dynamic* decomposition performed at run-time by the system itself; and we propose a continuum between them, with criteria for selecting a position along the continuum and for the temporal evolution of a system across it. The framework is intended as a design aid for researchers and practitioners building trustworthy, cognitively coherent multi-SLM systems.

Keywords

Multi-agent systems, Small Language Models, Task decomposition, Agentic AI, Software architecture, Compound AI systems

1. Introduction

The current discourse on agentic artificial intelligence (AI) is increasingly organised around a single architectural template: a large, general-purpose language model sits at the centre of the system and orchestrates every aspect of task execution, typically invoking external tools to extend its reach [1]. This template has produced genuine engineering progress, but it is a *choice*, not a necessity. We do not claim that mainstream practice ignores multiple agents: contemporary frameworks routinely decompose work across several agents and sub-agents organised into recognisable patterns—routing, parallel sectioning, and orchestrator–worker delegation in which a central model breaks a task down, dispatches the pieces, and synthesises the results [2]. What this practice typically leaves implicit is decomposition *as such*: the partition is asserted by a pipeline or a role chart rather than analysed as an architectural choice with its own trade-offs, and almost never with small, specialised models in view. A different template has long been available in the multi-agent systems (MAS) tradition, which holds that complex problems are often best addressed by distributed collections of specialised agents rather than by monolithic solvers [3, 4]. Until recently, the gap between that tradition and mainstream AI engineering was partly technological: there were no agent-level components with the kind of flexible, language-oriented competence that made coordinated problem solving attractive. The emergence of capable Small Language Models (SLMs)—compact foundation models, typically below ten

WOA 2026: 27th Workshop "From Objects to Agents", June 15–17, 2026, Salerno, Italy

*Corresponding author.

✉ pasquale.miranda314@gmail.com (P. Miranda); walter.balzano@unina.it (W. Balzano); alberto.moccardi@unina.it (A. Moccardi)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

billion parameters, trained on carefully curated data and often specialised on narrower families of tasks [5, 6]—changes that picture.

Belcak et al. [6] argue that SLMs are not merely a cost-reduction technique but are *structurally* better suited to a large fraction of agentic workloads, which consist of narrow, repetitive linguistic tasks for which a generalist model is over-dimensioned. The compound-AI-systems perspective [7] frames state-of-the-art results as emerging from assemblies of interacting components rather than from ever larger monolithic models. Together these positions converge on a classical MAS insight: when specialised components exist, the hard problem is no longer *how big* a single component should be, but *how the overall task is partitioned among components*.

The position we argue in this paper is that, for multi-agent systems of SLMs, *task decomposition* is the architectural lever that most directly shapes system behaviour, and that it is best treated as a first-class design concern rather than a by-product of role assignment. Communication protocols, memory models, orchestration patterns and trust mechanisms are all important, but they operate *on* a given decomposition. A poor decomposition cannot be rescued by sophisticated communication; a good decomposition simplifies every downstream concern, including verifiability and situation awareness which are central themes of current trustworthy-AI research. Accordingly, this paper concentrates on decomposition and deliberately leaves inter-agent communication outside its scope, returning to it only briefly as a complementary open direction in Section 7.

Our specific contributions are four. First, we *characterise* task decomposition as a problem with features that distinguish it both from classical MAS decomposition and from prompting strategies internal to a single language model (Section 3). Second, we articulate a framework for *static* decomposition, i.e. decomposition designed ahead of time by the system architect, covering separability criteria, recurring patterns, and conditions of applicability (Section 4). Third, we articulate a parallel framework for *dynamic* decomposition, in which the system itself produces a task-specific decomposition at run-time, covering components, mechanisms, challenges, and conditions of applicability (Section 5). Fourth, we propose a *continuum* between the two regimes, with hybrid configurations, selection criteria, and a notion of temporal evolution of a system along the continuum (Section 6). We close with a discussion of limitations and open problems (Section 7) and a brief conclusion (Section 8).

The target audience is twofold: MAS researchers engaging with language-model-based systems without abandoning the conceptual tools of the agent-oriented tradition, and practitioners who need a principled way to reason about the division of labour in systems built on multiple heterogeneous SLMs. Our tone is architectural rather than prescriptive: we describe trade-offs and choice points, and keep formalisation light.

2. Background and Related Work

Five bodies of work intersect the problem addressed in this paper. We summarise each, identify what it contributes to our framework, and flag what it leaves unresolved for the specific case of multi-SLM systems.

Classical task decomposition in MAS and AI planning. Hierarchical Task Networks (HTN), introduced in their modern form by Erol, Hendler and Nau [8, 9], formalise decomposition as the recursive refinement of abstract tasks into networks of more concrete ones, with complexity and expressiveness results that remain a foundational reference. Coalition formation for task allocation, as developed by Shehory and Kraus [10], studies how sets of agents organise themselves to cover tasks under resource and utility constraints. Belief–Desire–Intention (BDI) architectures [4] and behaviour-based control [11] supply complementary views on how agents deliberate and act. A common assumption runs through this literature: agent capabilities are *specifiable*—one can, at design time, characterise what an agent can do and with what quality, and reason about allocation on that basis. SLMs violate this assumption. Their capabilities are distributions of behaviour over input classes, not deterministic functions, and those distributions are only partially known in advance. Classical frameworks therefore

apply in spirit but require adaptation in substance. Recent applications continue to draw on the agent-oriented tradition in domains where coordination among specialised components is paramount, as in agent-based recommendation for cultural tours [12], which shows that the classical division-of-labour view remains productive well beyond its original planning context.

Compound AI systems and LLM-based agent frameworks. The compound-AI-systems framing [7] explicitly describes state-of-the-art AI artefacts as assemblies of multiple interacting components—models, retrievers, tools—rather than single models. Survey work on LLM-based agents [1] and on LLM-based multi-agent systems [13] maps a rapidly growing space of designs. On the practitioner side, frameworks such as AutoGen [14], MetaGPT [15] and CAMEL [16] provide infrastructure for orchestrating multiple LLM-powered agents. Within this literature, task decomposition is often present but rarely thematised: it surfaces as pipelines of fixed roles (e.g. product manager, architect, engineer in MetaGPT) or as coordinator–worker patterns whose structure is asserted rather than analysed. The design space of decomposition strategies and their trade-offs remains largely under-examined, particularly for the case of small, specialised models rather than a single large generalist.

Intra-model task decomposition. A distinct strand of work decomposes tasks *inside* a single language model. Chain-of-thought prompting [17] elicits intermediate reasoning steps; Tree-of-Thoughts [18] generalises this to branching search over reasoning states; least-to-most prompting [19] and Plan-and-Solve [20] explicitly split a problem into ordered sub-problems; ReAct [21] interleaves reasoning and tool actions. These techniques are powerful and influential, but they operate at the *prompt* level inside a single model. They do not address the question of how sub-tasks should be allocated to *different* specialised models with heterogeneous competences. Where such intra-model techniques inform our framework, we draw on them as primitives that dynamic decomposition components may themselves use (Section 5); they do not substitute for an architectural account.

Mixture-of-Experts. The sparsely-gated Mixture-of-Experts layer [22] introduces conditional computation: a gating network routes each input to a small subset of sub-networks. MoE has clear conceptual kinship with multi-SLM decomposition—both partition computation among specialists and both rely on routing—but a structural difference matters. MoE is a mechanism *internal* to a single trained model, end-to-end differentiable and operating at the granularity of token-level activations, whereas multi-SLM systems are assemblies of *independent services*, non-differentiable across boundaries and composed at the architectural level. MoE is a useful reference but does not supply the architectural language we need.

Service decomposition in software architecture. Outside AI, the decomposition of software systems into fine-grained services is a mature topic. Newman’s treatment of microservices [23] articulates criteria such as bounded contexts, information hiding, low coupling and high cohesion, together with guidance on splitting monoliths and reasoning about interface design. The analogy with multi-SLM systems is useful and partly exploited later in this paper. However, microservices assume *deterministic* services with specifiable interfaces and behaviours. SLMs are probabilistic components whose behaviour on a given input is a distribution, not a function; and natural-language interfaces between SLMs are looser than the typed contracts microservices presuppose. We therefore treat microservices wisdom as a source of useful heuristics to be reinterpreted, not as a framework to be imported wholesale.

Small language models as agentic components. The position that SLMs are not a stopgap but the appropriate default for much agentic work is argued at length by Belcak et al. [6], who contend that typical agentic workloads are narrow and repetitive enough that a generalist model is overdimensioned, and sketch a migration path from LLM- to SLM-based agents. Survey evidence on SLMs in agentic systems [24] corroborates this for schema- and API-constrained tasks—tool use, function calling, structured extraction—where compact models match or surpass larger ones at a fraction of the

cost, while delineating the residual regime (open-domain reasoning and long-horizon planning) where a larger model remains valuable. Two consequences matter for what follows. First, this is the literature that motivates studying decomposition specifically for small, specialised models rather than a single generalist. Second, the same work points beyond pure SLM ensembles toward *hybrid* arrangements in which an SLM-default system escalates to a stronger model on the sub-tasks that need it—a possibility our framework must accommodate rather than ignore (Sections 3 and 6).

Gap and positioning. Across these areas, task decomposition for multi-SLM systems has no dedicated conceptual treatment. Classical MAS planning and allocation assume specifiable capabilities; compound-AI and LLM-agent literatures describe specific systems rather than decomposition as an architectural dimension with its own trade-offs; intra-model prompting decomposes inside a single model, not across specialists; MoE works at a different level of abstraction; and microservices address deterministic services. The gap we address is precisely this one: an architectural framework that treats decomposition as a first-class concern in systems composed of heterogeneous, stochastic, language-oriented specialists.

3. Characterisation of the Problem

Before examining decomposition strategies, we need to say precisely what makes task decomposition for multi-SLM systems a problem of its own, not a mere special case of classical MAS planning or of intra-model prompting. Several features stand out. They are not independent axes to be optimised separately; they are interacting facets of a single underlying difficulty that any design decision must address jointly.

Stochastic, partially-specifiable capabilities. Classical approaches to task allocation assume that what an agent *can do* and *how well* are knowable at design time, at least up to declared uncertainty [10]. For an SLM this assumption does not hold in the strict form the classical theory needs. An SLM has a distribution of behaviours over classes of input: prompted appropriately, it performs some tasks reliably, others with qualified accuracy, others poorly, and for many inputs its behaviour is only partially characterisable from its published evaluations [6]. Let $\mathcal{I}$ denote the space of admissible inputs and $\mathcal{O}$ the space of possible outputs; the behaviour of a specialist s is best modelled as a conditional distribution $P_s(o \mid i)$ over $o \in \mathcal{O}$ for each $i \in \mathcal{I}$, with task-specific quality functionals $q_s(i)$ that are themselves uncertain. Decomposition must be defensible under this uncertainty, not against an idealised specification. Any allocation decision is therefore a bet on P_s restricted to the sub-task’s expected input distribution.

What is specific to small models. The previous feature holds for any stochastic agent, but SLMs sharpen it in a way that is specific to their scale and central to this paper. By construction, SLMs tend to lack several of the emergent capabilities on which larger models can be relied: extended multi-step reasoning, robust in-context learning from long or unfamiliar prompts, and reliable self-correction [6, 24]. On the narrow, schema- or API-constrained tasks that make up much agentic work, this is no handicap—a well-chosen SLM matches or beats a generalist at a fraction of the cost. The difficulty is that decomposition relocates exactly the capabilities SLMs are weakest at. Producing a good partition (the *decomposer*’s job) and reconciling partial outputs into a coherent whole (the *recomposer*’s job, discussed below) are open-ended reasoning tasks of precisely the kind a small model performs least reliably. The architectural implication is twofold. Whereas an LLM-centric system can decompose *inside* the same model that executes, a multi-SLM system cannot assume the orchestration roles are within reach of the same class of model as the workers; the decomposer and recomposer are the bottleneck of the regime. This makes *hybrid* arrangements a first-class option rather than an afterthought: a more capable model (a larger SLM or an LLM) can be reserved for decomposition, recomposition, and arbitration, while specialised SLMs execute the sub-tasks, with escalation to a stronger model on the inputs that warrant

it [24]. We return to this hybrid configuration as a position on the continuum in Section 6 and as a factor to be measured in Section 7.1.

Decomposition as an epistemic commitment. Decomposing a task embodies a structural hypothesis: that it *admits* a decomposition of a given shape. This is not neutral. A problem that appears to split cleanly into sub-problems *A* and *B* may carry cross-dependencies that make the split counter-productive—the quality of *A*’s output may depend on information surfaced only while solving *B*, or *B* may presuppose choices made within *A* that are invisible across the interface. Classical HTN planning makes this commitment explicit through method libraries and decomposition axioms [8, 9]; in multi-SLM systems it is often implicit, embedded in prompts, role definitions and routing rules, and therefore harder to audit. A central requirement for trustworthy multi-SLM design is to make the decomposition hypothesis itself inspectable.

The recomposition problem. Decomposing without a viable recomposition is an empty move. In classical MAS settings recomposition is frequently trivial—independent contributions accumulate in a shared environment or are aggregated by a simple utility function. For SLMs producing text, recomposition is itself a cognitive task: the outputs of specialists must be reconciled into a coherent whole that preserves semantic consistency, resolves ambiguities, and handles disagreements. Architecturally, the recomposition step is as much a first-class component of the system as the specialists themselves, and its design choices constrain which decompositions are feasible. A decomposition whose parts cannot be glued back together is not a decomposition; it is a partition.

A related observation bears on the scope of this paper. Every decomposition implicitly defines a *communication topology*—which specialists exchange information, in what direction, and at which points of the workflow—and although we leave inter-agent communication outside our scope (see Section 7), the coupling cannot be wished away: interface and context-passing costs feed back into the evaluation of the decomposition itself. We therefore treat communication cost as implicit in the notion of “minimal interfaces” (Section 4) rather than as a separate optimisation target.

Capability-dependent granularity. Granularity—how finely the task is split—does not admit an abstract optimum. Many small sub-tasks exploit specialisation but multiply coordination cost and the number of interfaces at which information may be lost; a few large sub-tasks reduce coordination but may force specialists to operate outside their zones of competence. The optimal granularity depends on the roster available: narrowly-scoped but highly competent SLMs reward fine-grained decomposition, whereas a few general-purpose SLMs gain little from finer partitioning beyond a point. Decomposition design and roster design are therefore *co-dependent*. This is the MAS counterpart of what the microservices literature calls the alignment between service boundaries and team or capability boundaries [23].

The static–dynamic tension. A final feature pervades the entire design space: the choice between fixing the decomposition at design time and producing it at run-time. Static decomposition yields predictability, auditability, and efficient operation at the cost of inflexibility in the face of novel tasks. Dynamic decomposition yields adaptivity at the cost of additional run-time machinery, consistency concerns, and a harder auditability story. This tension is not resolvable in the abstract: which regime is preferable depends on the domain’s rate of change, the variability of tasks, the maturity of the available specialists, and the operational constraints on verifiability and latency. Rather than treating the two regimes as alternatives, we argue in Section 6 that they form endpoints of a continuum along which real systems are positioned—and along which a given system may move as it matures.

3.1. Operational Criteria for Decomposition Quality

The features just described imply an operational standard for what counts as a *good* decomposition—a compact grid an architect or reviewer can apply to any candidate partition. We name it here and let

it recur in specialised form in the separability criteria of Section 4 and the challenges of Section 5. Five dimensions matter. *Fitness to specialist capabilities*: does each sub-task fall within the competence envelope P_s of its assigned specialist, rather than forcing it to the edge or outside? *Interface economy*: how much information must cross sub-task boundaries, and in what form—partitions that ship large context windows across interfaces carry hidden communication costs and are fragile under context-length limits. *Recomposition burden*: how expensive and how reliable is the step that glues partial outputs together, and does it merely relocate difficulty from the specialists into the recomposer? *Local evaluability*: can each sub-task’s quality be assessed in isolation, without running the full system? *Robustness under task variation*: does a small perturbation of the input produce a commensurately small change in the partition, or force a structurally different one? These are not quantitative metrics and they do not yield a total order—they routinely conflict—but they make explicit the trade-offs any choice of partition is making, and the rest of the paper applies this grid in the two regimes where those trade-offs are resolved.

4. Static Decomposition

In *static decomposition*, the division of a task into sub-tasks is fixed at design time. The architect inspects the problem domain, decides how to partition it, assigns sub-tasks to roles, binds roles to specialists, and configures a recomposition step. At run-time, the system follows this structure for every incoming request: no decision about *how* to split the task is made on the fly. MetaGPT [15], with its assembly-line of fixed roles, is an illustrative instance; so are most RAG-plus-agent pipelines, which realise a fixed partition between retrieval, generation, and post-processing. This section develops a framework for reasoning about static decomposition in multi-SLM systems along three dimensions: criteria of separability, recurring patterns, and conditions of applicability.

4.1. Criteria of Separability

A decomposition that is *valid*—structurally sound, independently of whether it is the best among alternatives—should satisfy five criteria. These criteria generalise classical MAS decomposition properties [8, 10] and the information-hiding principles of microservices design [23] to the specific case of stochastic language specialists.

Semantic separability. The sub-tasks must be distinct enough in meaning that they can be addressed in isolation. Superficial separability (two sub-tasks concern different sections of a document) does not imply semantic separability (understanding either section may require context established by the other). A useful test is whether a competent human specialist could tackle one sub-task without needing to perform the others first.

Minimal interfaces. The information that must pass between sub-tasks should be limited and well defined. An interface that requires passing arbitrarily large context or the full conversational history across specialists is not a decomposition; it is distribution in name only, with all sub-tasks effectively sharing the same state. In a multi-SLM setting, interface minimality is directly tied to context-window budgets and, through them, to cost.

Feasible recomposition. There must exist an efficient recomposition procedure that combines specialist outputs into a coherent overall result. Feasibility has two faces: computational (the recomposition must not dominate the cost of the system) and cognitive (it must handle the kinds of inconsistencies SLMs typically produce, including factual conflicts, stylistic mismatches, and partial answers).

Coherent quality criteria. It must be possible, at least approximately, to evaluate the quality of each sub-task in isolation. If only the end-to-end output is judgeable, then neither local improvement nor meaningful monitoring of individual specialists is possible; the decomposition is then architecturally brittle even when it works, because failures cannot be localised.

Stability under input variation. Small variations in the overall input should not force structural changes in the decomposition. If a minor reformulation of a user request requires a different partition

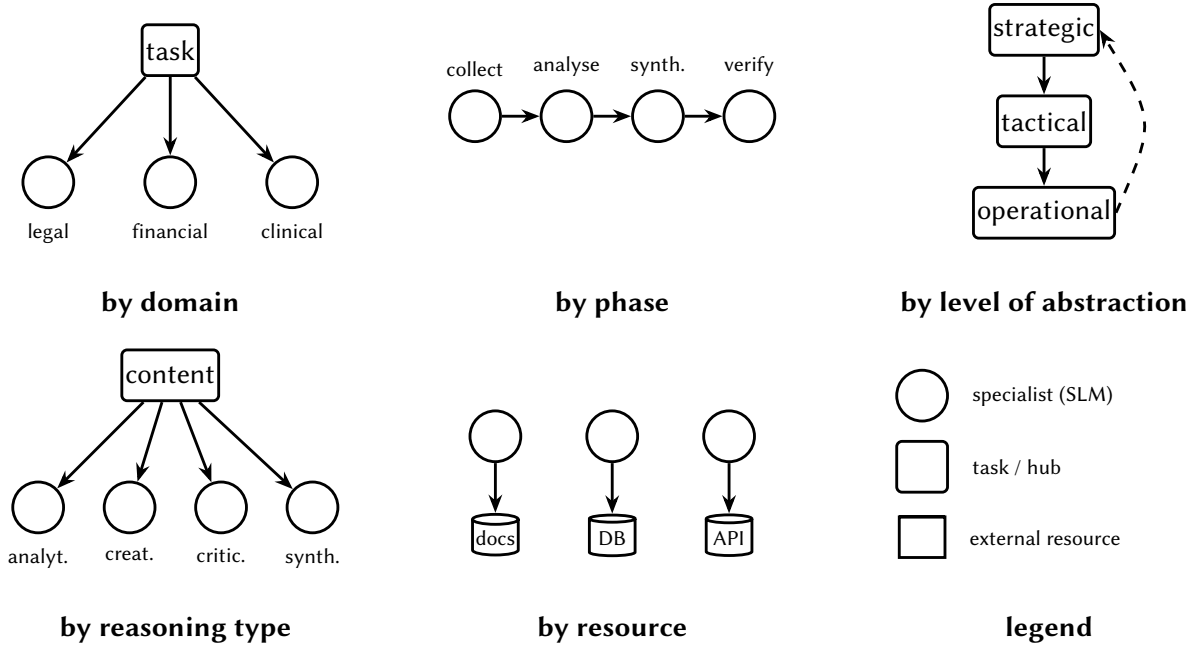


Figure 1: The five recurring patterns of static decomposition introduced in this section. Each panel shows the characteristic topology induced by the pattern: specialists fan out from a shared task (by domain, by reasoning type), form a pipeline (by phase), are arranged in a hierarchy with feedback (by level of abstraction), or are bound to distinct external resources (by resource). Real systems frequently combine two or more patterns at different levels of the design.

of sub-tasks, the design has captured a single instance rather than a class. Stability is what makes a static decomposition worth fixing in the first place.

4.2. Recurring Patterns

We observe five recurring patterns of static decomposition in the literature and in practice, summarised in Figure 1. The patterns are not mutually exclusive: real systems frequently combine two or more (for example, a domain partition at the top level and a phase partition within each domain).

Decomposition by domain. Sub-tasks are separated according to subject matter: legal analysis, financial analysis, clinical interpretation, and so on. Each sub-task is assigned to a specialist trained or fine-tuned for that domain. The strength of the pattern is the direct exploitation of domain-specific fine-tuning, which SLMs support particularly well; its weakness is cross-domain tasks, which force either a dominant specialist to over-reach or a recomposition step that must itself carry domain knowledge.

Decomposition by phase. Sub-tasks are separated according to their position in a workflow: collection, analysis, synthesis, verification. The system realises a pipeline of specialists, each operating on the output of the previous stage. This pattern is common in compound AI systems [7] and in software engineering multi-agent systems [15], and it inherits the well-understood properties of pipelines (clarity, easy monitoring, susceptibility to error propagation).

Decomposition by level of abstraction. Specialists operate at different abstraction levels on the same problem: strategic, tactical, operational. Upper layers set constraints that lower layers must respect; lower layers surface facts that may trigger strategic revisions. The hierarchical organisation is reminiscent of both subsumption architectures [11] and hierarchical planning in the BDI tradition [4]. For multi-SLM systems the pattern offers a natural path to inspectability: high-level decisions can be audited independently of low-level details.

Decomposition by reasoning type. Specialists are distinguished by the kind of reasoning they perform on the same content: analytical decomposition, creative generation, critical evaluation, synthetic summarisation. Rather than splitting the material, the system splits the cognitive stance taken toward

it. The pattern has obvious kinship with Tree-of-Thoughts-style exploration [18] but at the inter-agent level: different specialists explore different stances and a recomposition step reconciles them.

Decomposition by resource. Sub-tasks are separated according to the external resources they access: a specialist with read access to a document corpus, another with access to a structured database, another with access to a web API. The separation reflects authorisation and safety boundaries as much as competence, and it intersects cleanly with trustworthy-AI concerns about controllable access to information. The pattern thus connects to a long-standing line of work on fine-grained access control over structured information, exemplified by semantic approaches to e-health document protection [25], where the unit of separation is dictated by the sensitivity of the underlying resource rather than by competence alone. While this pattern partly overlaps with decomposition by domain (resources and domains often correlate), the distinction matters: the *reason* for the split is different, and so are the failure modes—mis-routing, access errors, and silent fall-backs—that the architecture must handle.

4.3. Trade-offs in Choosing a Static Decomposition

Three trade-offs cut across all patterns. *Fine versus coarse*: more specialists yield more specialisation at the cost of more interfaces and coordination overhead. *Hierarchical versus flat*: a hierarchy gives control and clear lines of responsibility but adds layers; flat organisations are flexible but require peer-level coordination, which SLMs do not naturally support. *Parallel versus sequential*: parallelism improves latency but requires the sub-tasks to be genuinely independent, whereas sequential flows admit dependencies at the cost of longer critical paths and more points of error propagation.

None of these trade-offs has a universal optimum. Their resolution depends on the operational profile of the system—its latency budget, its tolerance for variance in output, its verifiability requirements—and on the roster of specialists, which together determine where additional specialisation yields value.

4.4. When Static Decomposition Is Appropriate

Static decomposition is the right default when several conditions hold: the problem domain is well understood and changes slowly; the tasks the system handles are repetitive or have stable structure; performance must be predictable and the cost of coordination must be amortised over many requests; verifiability and auditability are first-order requirements, as is often the case in regulated settings aligned with the trustworthy-AI concerns of WOA 2026—a concern that resonates with work on the formal verification and validation of compliance-bearing artefacts in regulated environments [26], where a fixed architectural skeleton is precisely what makes systematic checking possible; and operational conditions are stable enough that a fixed structure will remain fit for purpose over meaningful time horizons. Under these conditions, the rigidity of the static regime is a feature: it makes the system’s behaviour inspectable, its failures localisable, and its evolution controllable. The next section turns to the complementary case in which these conditions do not hold.

5. Dynamic Decomposition

In *dynamic decomposition*, the system does not carry a fixed partition of the task. Upon receiving a request, a component of the system analyses the incoming task and produces an *ad hoc* decomposition for that request. Specialists are engaged on the basis of the produced decomposition rather than on pre-assigned roles. The recomposition step is similarly task-specific. This regime is architecturally richer than the static one, more demanding to implement correctly, and, we argue, closer to the research frontier of multi-SLM systems. This section articulates its components, the conditions that trigger decomposition, the mechanisms by which decompositions are produced, the challenges specific to the regime, and the conditions under which it is appropriate.

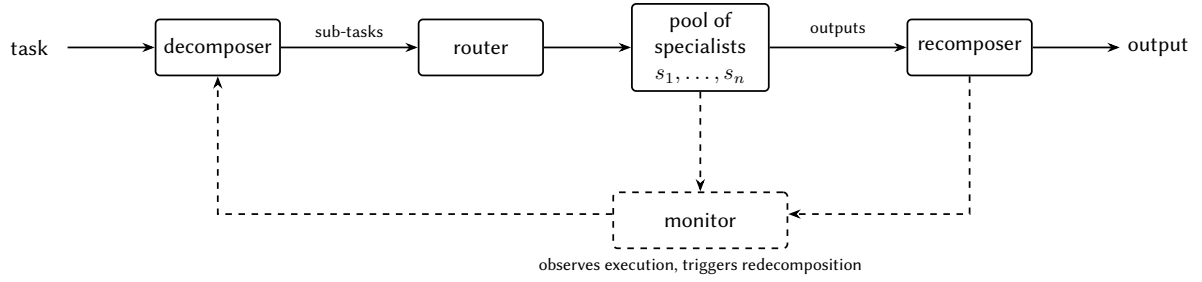


Figure 2: Architecture of a dynamic-decomposition system. Solid arrows carry task data; dashed arrows carry control and feedback signals. Specialists in the pool produce partial outputs that the recomposer aggregates. The monitor observes specialist outputs and the recombination step, and can trigger the decomposer to produce a revised partition.

5.1. Components of a Dynamic-Decomposition System

Five roles structure a dynamic-decomposition system, arranged as shown in Figure 2. Their implementations differ widely across systems, but the roles themselves are recognisable in any realisation of the regime.

The *decomposer* is the component that produces the decomposition. It may be implemented as a dedicated SLM prompted to propose a partition, as a more structured mechanism such as a template instantiator guided by a classifier, or as a hybrid in which an SLM proposes a decomposition and a symbolic component validates it. The decomposer is the single most consequential element of the regime: the quality of the partition it produces caps the quality of everything downstream.

The *pool of specialists* is the set of SLMs—possibly heterogeneous in size, fine-tuning, and modality—from which the decomposer may draw. The pool is typically described through metadata (capability tags, indicative quality levels on reference tasks, resource profile) that the decomposer and router can consult.

The *router* associates each sub-task in the proposed decomposition with a specialist in the pool, on the basis of capability metadata and current load. The router is where the conditional computation analogy with Mixture-of-Experts [22] becomes most tangible, with the essential difference that routing here happens at the granularity of independent services rather than differentiable sub-networks.

The *recomposer* aggregates the outputs of specialists into a coherent overall result. As noted in Section 3, for language-producing specialists this step is itself a cognitive task—often itself implemented by an SLM—and its design constrains which decompositions are feasible.

The *monitor* observes the system’s behaviour at run-time and can trigger revisions of the decomposition. Its function is both operational (detecting failures, degradations, loops) and architectural (providing the feedback signal that, in principle, allows a dynamic system to improve its decompositions over time).

5.2. Trigger Criteria

A purely static system needs no trigger: the decomposition is always the same. A dynamic system must decide when to (re)decompose. Four triggers recur across designs. Decomposition on the *arrival of every new task* yields the most adaptive but also the most expensive regime, since each request incurs the cost of producing and validating a partition. Decomposition on a *detected change of task class* amortises the cost by reusing a partition across similar tasks, with a classifier— sometimes an SLM itself—deciding when a new class has been entered. Decomposition on *failure or degradation* leaves the partition stable while things work and invokes the decomposer as a recovery mechanism, which requires that the monitor be reliable enough to justify not redecomposing whenever things are merely slow. Decomposition on the *availability of new specialists* reflects a longer-horizon rationale: when the pool changes, the prior partition may no longer be the best exploitation of available capability.

These triggers are not mutually exclusive; most deployed systems combine them. The architectural

commitment a designer makes is not *whether* to allow dynamic decomposition, but *which of these triggers* to wire into the system and with what priority.

5.3. Mechanisms for Producing a Decomposition

Five mechanisms cover most observed approaches. We present them in increasing order of architectural sophistication.

Prompted decomposition. The decomposer receives the task and a structured prompt that elicits a decomposition, often in the form of a list of sub-tasks with brief specifications. This is the direct multi-agent analogue of intra-model decomposition techniques such as least-to-most [19] and Plan-and-Solve [20], with the crucial difference that the resulting sub-tasks are dispatched to distinct specialists rather than resolved by the same model. The mechanism is simple and easy to deploy; its ceiling is set by the decomposer’s own decomposition competence, which can be surprisingly uneven.

Template-based decomposition. A library of decomposition templates is maintained for recognised task classes, and at run-time the decomposer selects and instantiates a template rather than generating a decomposition from scratch. The approach trades expressiveness for predictability: within the coverage of the library it approximates the behaviour of a well-designed static system, while outside it falls back to a less specialised template or to a fully generative decomposer. In practice, template-based decomposition is how many production systems realise the promise of dynamic decomposition while retaining auditability.

Decomposition by analogy. The system retains a memory of past decompositions together with the outcomes they achieved, and at run-time searches this memory for tasks similar to the incoming one, adapting the retrieved decomposition rather than constructing it anew. The mechanism has obvious kinship with case-based reasoning in classical AI, and it offers a natural integration point for learning: the memory can be curated, pruned, and reweighted as outcomes accumulate. For SLM systems, decomposition by analogy may be the most promising path to a system that genuinely improves its partitions over time without requiring full retraining of any component.

Iterative decomposition. A coarse initial partition is produced first, executed, and evaluated; if the result is unsatisfactory, the decomposition is refined and re-executed. The mechanism is expensive and introduces the risk of non-terminating loops (see below), but it captures a pragmatic insight: for many tasks, the right decomposition is not evident until one has attempted a first solution. Iterative decomposition can be viewed as the multi-agent counterpart of Tree-of-Thoughts [18], where the exploration happens over candidate *partitions of the task* rather than over reasoning paths inside a single model.

Collaborative decomposition. Several SLMs propose competing decompositions and an arbiter selects or merges them. Conceptually this is a multi-agent-debate pattern [16, 14] applied to the decomposition step itself, and it can produce higher quality partitions than any single proposer at the cost of additional coordination. Collaborative decomposition is best suited to high-stakes tasks where the cost of a poor partition far outweighs the cost of running several proposers.

These mechanisms can be composed. A system may, for example, first try retrieval (by analogy), fall back to template instantiation if no close match is found, and resort to a prompted decomposer only for genuinely novel tasks. Designing such layered pipelines is where the architectural work of dynamic decomposition largely sits.

5.4. Challenges Specific to the Regime

Dynamic decomposition introduces five classes of difficulty beyond those of the static case. *Quality of the decomposition:* a stochastic decomposer can produce structurally unsound partitions, and detecting this is non-trivial because the defect may surface only in recomposition. *Coordination cost:* producing a partition at run-time adds latency and compute that must be justified by the gain in fit. *Consistency:* two closely related tasks may receive meaningfully different decompositions, complicating reproducibility and comparison—precisely what operators of trustworthy systems care about. *Auditability:* the

decomposition trace is part of the system's behaviour and must be recorded, explained, and justified post hoc. *Loops and regressions*: a monitor that redecomposes on detected failure can loop if the new partition is no better, or progressively degrade the system if each is slightly worse than the last; robust operation requires explicit convergence criteria and hard bounds on redecomposition attempts.

5.5. When Dynamic Decomposition Is Appropriate

Dynamic decomposition is justified when the domain is heterogeneous and tasks have highly variable structure; when novel tasks are frequent and poorly covered by any pre-designed partition; when the available roster of specialists is itself heterogeneous enough that the optimal allocation genuinely varies from task to task; when flexibility is worth more than the additional coordination cost; and when the monitoring and fall-back machinery is robust enough to keep the regime safe in production. In the absence of such conditions, the static regime is preferable, if only because it is easier to operate and easier to trust. The interesting observation, however, is that most real systems satisfy some of these conditions but not all, and therefore sit neither at the purely static nor the purely dynamic extreme. The next section takes this observation seriously.

6. The Static–Dynamic Continuum

Presenting static and dynamic decomposition as clean alternatives is a useful expository device, but it misdescribes the systems that are actually deployed. Real multi-SLM systems typically combine elements of both: they carry some structure fixed at design time and produce other structure at run-time. Rather than treating this as a compromise between two ideals, we propose to treat the whole design space as a *continuum* with the pure regimes at its endpoints. This framing has three benefits. It matches observed practice. It makes visible intermediate configurations that are often invented *ad hoc* but deserve to be recognised as design patterns in their own right. And it exposes a *temporal* dimension: systems can move along the continuum as they mature.

6.1. Hybrid Configurations

Several intermediate configurations recur. The first four can be understood as well-defined positions on the continuum, closer to the static pole or to the dynamic one, depending on which aspects of the decomposition are fixed and which are variable; a fifth, hybrid orchestration, cuts across the continuum on a separate, capability-based axis.

Static structure with dynamic parameters. The decomposition's shape is fixed—which macro-phases exist, which specialist roles they involve—but run-time parameters (which model serves a role, how much budget each phase receives, whether an optional verification specialist is invoked) are chosen dynamically.

Template-based with adaptive selection. A library of decompositions is maintained at design time, and the system selects among them at run-time based on features of the incoming task. This is the position occupied by many production systems: the decomposer never invents a partition, but the space of partitions is rich enough that system behaviour varies meaningfully with input.

Hybrid hierarchy. The top-level decomposition is static—the same high-level phases are always present—while the decomposition of each phase is dynamic. This matches hierarchical-planning practices from the classical literature [8, 4], re-interpreted with SLMs as specialists, and combines the auditability of a fixed skeleton with the adaptivity of local reconfiguration.

Dynamic with memory. The decomposition is produced at run-time but informed by retained outcomes of past decompositions, so successive partitions tend toward stability for recurring task classes. Such systems drift toward templates over time without requiring the templates to be designed explicitly—appealing for domains whose task distribution is not fully known in advance.

Dynamic under static constraints. The decomposition is produced at run-time but must satisfy design-time invariants: certain specialists must always be consulted, certain verification steps must always run,

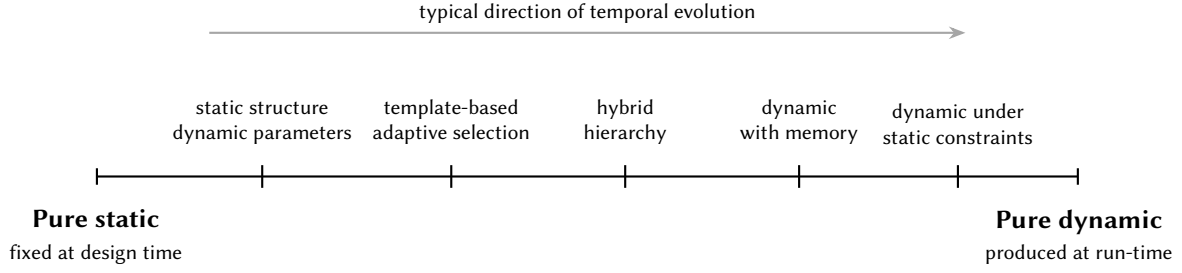


Figure 3: The static–dynamic continuum for task decomposition in multi-SLM systems. Five hybrid configurations occupy intermediate positions between the pure regimes. The top arrow indicates the typical direction of architectural maturation: systems tend to start nearer the static pole and move toward dynamic configurations as the domain is better understood and specialists mature.

certain interface contracts must be honoured regardless of the rest of the partition. Constraints play the role of a safety envelope around an otherwise dynamic regime, and they are where trustworthy-AI concerns [7, 6] most naturally attach.

Hybrid orchestration. A capability axis cuts across the four configurations above. Rather than fixing *when* the partition is decided, it fixes *who* decides it: a more capable model—a larger SLM or an LLM—takes the orchestration roles (decomposition, recomposition, arbitration) that the previous section identified as the weak point of a pure SLM ensemble, while specialised SLMs execute the sub-tasks, with selective escalation to the stronger model on inputs that warrant it [24]. The orchestrator–worker pattern of mainstream practice [2] is the limiting case in which orchestration is dynamic; pairing a capable orchestrator with a fixed skeleton yields a more auditable variant. Hybrid orchestration is largely orthogonal to the static–dynamic placement and can be combined with any of the four configurations above; it is the most direct answer to the question of how to exploit small specialists without inheriting their limitations at the points where reasoning is most demanded.

Figure 3 places these hybrid configurations on the static–dynamic continuum and indicates the typical direction of temporal evolution discussed in Section 6.3.

6.2. Criteria for Positioning a System

Six factors push a system toward one pole or the other. *Domain stability* pushes toward the static pole: if the problem space rarely changes, there is little to be gained from run-time flexibility. *Task variability* pushes toward the dynamic pole: if every request is structurally different, a fixed partition cannot serve. *Auditability requirements* pull toward the static pole, because a fixed structure is easier to inspect and justify. *Adaptivity requirements* pull toward the dynamic pole, by symmetric reasoning. *Maturity of available specialists* is subtler: the more capable and the better-characterised the specialists, the more one can trust a dynamic regime to exploit them sensibly; less mature pools favour static configurations in which the architect compensates for specialists’ limitations by hand. *Volume and latency budgets* finally push toward the static pole, because producing a decomposition at run-time is not free: systems with tight latency or high throughput tend to amortise the decomposition step across requests.

No single factor dominates. A well-designed system reads the weights of these factors in its specific context and lands at a position on the continuum that reflects the trade-off the operator is willing to make. Table 1 collects the six factors in a compact decision grid that practitioners can use as a first reading of where a system should sit.

6.3. Temporal Evolution

The final observation is that a system’s position on the continuum is not a one-time choice. Systems evolve, and a sensible evolution pattern runs from the static pole toward the dynamic one.

A young system—early in the life of a deployment, with immature specialists and an under-characterised task distribution—is typically better served by a static decomposition, which is easier to

Table 1

Decision grid for positioning a multi-SLM system on the static–dynamic continuum. The table consolidates the six factors discussed above; in practice several factors typically apply at once, and the resulting position reflects their combined weight rather than any single dominant signal.

Condition	Pushes toward	Typical configuration
Stable, well-understood domain	static	fixed pipeline of specialists
High task variability, frequent novelty	dynamic	prompted or iterative decomposer over a diverse pool
Strict auditability requirements	static	static structure, or dynamic under static constraints
Strong adaptivity requirements	dynamic	template-based or memory-backed dynamic decomposition
Immature specialist pool	static	hand-crafted partition compensating for specialist limits
Heterogeneous, well-characterised pool	dynamic or hybrid	learned routing, analogy-based decomposition with memory
Tight latency or high-volume throughput	static	static structure, possibly with dynamic parameters only

build, debug, and trust. As the system matures, two things happen: the operator learns which parts of the domain are variable enough to warrant adaptivity, and the specialists themselves become more predictable through fine-tuning, improved prompts, and accumulated evaluation. Both developments expand the space in which dynamic decomposition is safe. It therefore makes sense to *start* static and move toward dynamic components selectively, replacing fixed structure with run-time flexibility where the evidence justifies it—not a one-off migration but a pattern of architectural maturation.

We do not claim that every system should end up at the dynamic pole. For many domains, the static or near-static configurations will remain the correct choice indefinitely, for entirely sound operational reasons. What we do claim is that the direction of research progress—and the direction in which the field is likely to find architecturally interesting problems—points toward dynamic decomposition, understood as the capacity of a multi-SLM system to produce increasingly effective partitions of its tasks with decreasing direct architect intervention. This orientation aligns with the compound-AI-systems agenda [7] and with the position that SLMs are structurally suited to agentic workloads [6]; it is also where the WOA 2026 theme of trustworthy autonomous systems meets its sharpest architectural question: how much autonomy over its own structure should a multi-agent system have?

7. Discussion

The framework developed in this paper is conceptual, and several of its commitments deserve explicit scrutiny.

Formalisation. We have kept formalisation light, introducing notation only where it sharpened a distinction. A fuller formal treatment—of separability criteria as properties of task graphs, of decompositions as mappings from tasks to partitions with quantifiable coupling, of the continuum as a parameterised family of systems—is a natural next step, particularly for connecting the framework to verification and reproducibility tooling.

Scope: the communication question. We deliberately placed inter-agent communication outside the scope of this paper. This is a strategic simplification, not a claim of independence. A decomposition determines *what* must be communicated between specialists; a communication protocol determines *how*. The two questions meet at every interface, and a full account of multi-SLM systems needs both. Our wager is that treating decomposition first—as the architectural primitive that shapes the communication problem—produces cleaner analyses than treating both simultaneously. The natural companion to this paper is therefore a treatment of agent communication for multi-SLM systems that takes our

decomposition framework as its input.

Empirical grounding. The framework is derived from analysis of the literature and of observed patterns in deployed systems rather than from controlled experiments. This is appropriate for a conceptual contribution but creates an obligation: the categories we propose must be tested against real systems, and refined where the categories do not cut the space cleanly. We expect the list of patterns in Sections 4 and 5 to evolve with the field, and indeed we would be surprised if it did not.

7.1. How the Framework Could Be Evaluated

Although this paper does not report experiments, the framework is not meant to stay purely speculative. A minimal empirical programme that would begin to test it can be sketched along two axes: metrics and scenarios.

Metrics. The quality grid of Section 3.1 translates directly into four measurable dimensions. (i) *Task quality*: standard end-to-end performance on the target task, measured against whatever reference is appropriate to the domain (exact match, rouge, human rating, task-specific scorers). (ii) *Cost and latency*: total token usage across specialists, wall-clock latency to completion, and—where applicable—the overhead specifically attributable to decomposition and recomposition, separated from the cost of the specialists themselves. (iii) *Auditability*: the fraction of system outputs that can be traced to a specific specialist and a specific sub-task, and whether the decomposition used for a given request can be reconstructed from logs. (iv) *Stability across similar tasks*: the extent to which structurally similar inputs receive similar decompositions, which for dynamic systems is a non-trivial property and directly relevant to reproducibility.

Scenarios and instruments. The case studies need not be built from scratch: existing agentic benchmarks already supply the two contrasting regimes. A *stable, repetitive* workload—structurally homogeneous requests such as schema-constrained extraction, function calling, or document classification, evaluable on public suites for tool use and structured output (of the kind surveyed in [24])—is where a well-designed static decomposition should dominate on cost and auditability while losing little on task quality. A *heterogeneous, novelty-rich* workload—mixed analytical and generative requests drawn from a shifting distribution—is where a dynamic or hybrid decomposition should recover task-quality ground a static system leaves on the table, at the price of higher decomposition overhead. The specialist roster can be drawn from openly available SLM families, and a third, *hybrid* arm—a capable orchestrator over small workers, with escalation—would test whether the configuration of Section 6 buys task quality on the novelty-rich workload without surrendering the cost profile that makes SLMs attractive.

A study of this kind would not validate the framework in a strong sense, but it would provide the first empirical grounding for its central claims: that the static–dynamic distinction tracks a real performance trade-off, that the continuum configurations of Section 6 correspond to distinguishable operating points, and that the quality grid captures properties that actually matter under varying task distributions. We intend this as a concrete agenda rather than a decorative promise.

Open problems. Three stand out. First, the *learned decomposer*: under what conditions does decomposition by analogy, combined with outcome-weighted memory, produce a system whose partitions are demonstrably better than those of a fixed architect design, and how is “better” operationalised? Second, *safe redecomposition*: what are the architectural guarantees that rule out the loops and regressions of Section 5, and how can those guarantees be surfaced to operators in regulated settings? Third, *co-design of roster and decomposition*: given the co-dependence noted in Section 3, can the two be optimised jointly, or does one always lead the other in practice?

Limits. The framework is a design aid, not a design algorithm. It will not tell an architect which decomposition to adopt for a given problem; it will tell her which questions to ask, which trade-offs she is making, and in which region of the design space she is operating. We think this is the right level of abstraction for a position on an architectural dimension that the field has not yet named precisely, but readers who expect a decision procedure will be disappointed.

8. Conclusion

Task decomposition is the architectural dimension that most directly shapes the behaviour of multi-agent systems of Small Language Models. We have argued that it deserves to be analysed as a first-class design concern, distinct from classical MAS task allocation on the one hand and from intra-model prompting strategies on the other. We characterised the problem along five axes that reflect what is specific to stochastic, language-oriented specialists; we developed two complementary frameworks, for static and for dynamic decomposition, each articulated through criteria, patterns, mechanisms, and conditions of applicability; and we argued that the two regimes are best understood as endpoints of a continuum, along which well-designed systems move as the domain becomes better understood and the roster of specialists matures.

The contribution is conceptual and the framing is deliberately restrained: decomposition is a lever, not a silver bullet, and the directions the framework points toward—learned decomposers, safer redecomposition, joint roster–decomposition design—are research programmes rather than finished answers. If the paper achieves nothing else, we hope it makes decomposition choices explicit in the design discussions where they are currently buried.

Acknowledgments

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude (Anthropic) in order to: assist with literature search and verification of bibliographic references; draft and refine English prose; and review structural consistency across sections. After using this tool, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication’s content.

References

- [1] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, Q. Zhang, W. Qin, Y. Zheng, X. Qiu, X. Huang, T. Gui, The rise and potential of large language model based agents: A survey, arXiv preprint arXiv:2309.07864 (2023). [arXiv: 2309. 07864](https://arxiv.org/abs/2309.07864).
- [2] E. Schluntz, B. Zhang, Building effective agents, Anthropic Engineering Blog, 2024. URL: <https://www.anthropic.com/engineering/building-effective-agents>, accessed: 2026-06-06.
- [3] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed., John Wiley & Sons, Chichester, 2009.
- [4] A. S. Rao, M. P. Georgeff, BDI agents: From theory to practice, in: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), AAAI Press, 1995, pp. 312–319.
- [5] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, et al., Phi-3 technical report: A highly capable language model locally on your phone, arXiv preprint arXiv:2404.14219 (2024). [arXiv: 2404. 14219](https://arxiv.org/abs/2404.14219).
- [6] P. Belcak, G. Heinrich, S. Diao, Y. Fu, X. Dong, S. Muralidharan, Y. C. Lin, P. Molchanov, Small language models are the future of agentic AI, arXiv preprint arXiv:2506.02153 (2025). [arXiv: 2506. 02153](https://arxiv.org/abs/2506.02153).
- [7] M. Zaharia, O. Khattab, L. Chen, J. Q. Davis, H. Miller, C. Potts, J. Zou, M. Carbin, J. Frankle, N. Rao, A. Ghodsi, The shift from models to compound AI systems, Berkeley Artificial Intelligence Research Blog, 2024. URL: <https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/>, accessed: 2026-04-18.
- [8] K. Erol, J. Hendler, D. S. Nau, HTN planning: Complexity and expressivity, in: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), AAAI Press, 1994, pp. 1123–1128.

- [9] K. Erol, J. Hendler, D. S. Nau, Complexity results for HTN planning, *Annals of Mathematics and Artificial Intelligence* 18 (1996) 69–93. doi:10.1007/BF02136175.
- [10] O. Shehory, S. Kraus, Methods for task allocation via agent coalition formation, *Artificial Intelligence* 101 (1998) 165–200. doi:10.1016/S0004-3702(98)00045-9.
- [11] R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* 2 (1986) 14–23. doi:10.1109/JRA.1986.1087032.
- [12] F. Amato, F. Moscato, V. Moscato, F. Pascale, A. Picariello, An agent-based approach for recommending cultural tours, *Pattern Recognition Letters* 131 (2020) 341–347. doi:10.1016/j.patrec.2020.01.005.
- [13] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, X. Zhang, Large language model based multi-agents: A survey of progress and challenges, in: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, 2024, pp. 8048–8057. doi:10.24963/ijcai.2024/890.
- [14] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, C. Wang, AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework, *arXiv preprint arXiv:2308.08155* (2023). arXiv:2308.08155.
- [15] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, J. Schmidhuber, MetaGPT: Meta programming for a multi-agent collaborative framework, in: *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. arXiv:2308.00352.
- [16] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, B. Ghanem, CAMEL: Communicative agents for “mind” exploration of large language model society, in: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. arXiv:2303.17760.
- [17] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022. arXiv:2201.11903.
- [18] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, in: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. arXiv:2305.10601.
- [19] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le, E. Chi, Least-to-most prompting enables complex reasoning in large language models, in: *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023. arXiv:2205.10625.
- [20] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, E.-P. Lim, Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, in: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023)*, 2023, pp. 2609–2634. doi:10.18653/v1/2023.acl-long.147.
- [21] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing reasoning and acting in language models, in: *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023. arXiv:2210.03629.
- [22] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, J. Dean, Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, in: *5th International Conference on Learning Representations (ICLR 2017)*, 2017. arXiv:1701.06538.
- [23] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed., O’Reilly Media, 2021.
- [24] R. Sharma, M. Mehta, Small language models for agentic systems: A survey of architectures, capabilities, and deployment trade-offs, *arXiv preprint arXiv:2510.03847* (2025). arXiv:2510.03847.
- [25] F. Amato, V. Casola, N. Mazzocca, S. Romano, A semantic approach for fine-grain access control of e-health documents, *Logic Journal of the IGPL* 21 (2013) 692–701. doi:10.1093/jigpal/jzs027.
- [26] F. Amato, G. Cozzolino, F. Moscato, V. Moscato, F. Xhafa, A model for verification and validation of law compliance of smart contracts in IoT environment, *IEEE Transactions on Industrial Informatics* 17 (2021) 7752–7759. doi:10.1109/TII.2021.3057595.