ВЕРИФІКАЦІЙНИЙ ШАР В АРХІТЕКТУРАХ СИСТЕМ НА ОСНОВІ LLM-АГЕНТІВ: СИСТЕМАТИЗАЦІЯ ЗА ДЖЕРЕЛОМ ІСТИНИ

Киселевич Володимир Володимирович
асистент кафедри комп'ютерних наук та інформаційних технологій
Житомирський державний університет імені Івана Франка

Програмні системи на основі великих мовних моделей (LLM) дедалі частіше виконують не дорадчу, а виконавчу роль: агент не лише пропонує варіант дії, а й безпосередньо змінює стан зовнішніх систем, зокрема записує дані, надсилає запити, змінює програмний код. Ядро такої системи є ймовірнісним, тому помилковий вихід агента не є винятковою подією; його слід вважати штатним режимом роботи, який архітектура має передбачати. Механізм, що перехоплює вихід агента до виконання дії, у літературі називають верифікаційним шаром.

Проблема полягає в тому, що поняття верифікації в агентних системах уживають надто широко. Ним позначають і повторний запит до тієї самої моделі з проханням перевірити власну відповідь, і запуск модульних тестів, і формальне доведення властивостей згенерованого коду. Ці механізми мають принципово різну доказову силу, проте в описах архітектур їх подають як однорідні складники. Практичний наслідок такий: із твердження про наявність верифікації не випливає, що система має хоча б одне джерело істини, незалежне від самої моделі.

Аналіз останніх досліджень свідчить про таке. Механізми самокорекції опрацьовано найповніше: у праці [1] запропоновано ітеративне вдосконалення відповіді за власним відгуком моделі, а в праці [2] описано механізм самокритики з накопиченням досвіду в мовній формі. Проте в праці [3] показано, що моделі не здатні надійно виправляти власні міркування без зовнішнього сигналу, а в праці [4] встановлено, що самовдосконалення здатне посилювати власне зміщення моделі. Протилежний полюс представлено в праці [5], де безпеку агента забезпечують формальною перевіркою згенерованого коду перед виконанням. Роботи 2026 року засвідчують зміщення уваги до контролю під час виконання: у праці [6] описано монітор із формальною політикою втручання, що передбачає чотири дії: дозвіл, блокування, запит підтвердження та запит перегляду, у праці [7] описано поведінкові контракти агента з формальною специфікацією та примусовим виконанням. Наявні оглядові праці з надійності агентних систем [8] систематизують загрози та контрзаходи за модулями агента, тобто за тим, де застосовано захист. Натомість упорядкування механізмів перевірки за тим, звідки система дістає підставу вважати вихід правильним, у розглянутих джерелах не виявлено. Саме такий погляд і пропонується нижче.

Мета дослідження полягає в тому, щоб обґрунтувати виокремлення верифікації в самостійний архітектурний шар та систематизувати її механізми за

джерелом істини, тобто за тим, звідки система дістає підставу вважати вихід агента правильним.

Верифікацію доцільно розглядати як окремий шар, а не як деталь реалізації агента, з двох причин. По-перше, вона має власний предмет контролю: агент відповідає за породження варіанта дії, а верифікаційний шар відповідає за допуск цього варіанта до виконання. По-друге, вона змінюється незалежно від моделі: ту саму модель можна розгорнути і без перевірки, і з формальним доведенням властивостей виходу, причому гарантії системи будуть різними за незмінного ядра.

Пропонована ознака систематизації спирається на класичне поняття інженерії програмного забезпечення. Проблему оракула, тобто відсутність незалежного джерела, здатного відрізнити правильний результат від помилкового, докладно опрацьовано в праці [9]. Верифікація агентної системи є окремим випадком цієї проблеми, ускладненим тим, що перевіряють вихід імовірнісного породжувача. Тому механізми верифікації доцільно впорядкувати за джерелом істини, тобто за інстанцією, чиє рішення система бере за підставу допустити дію. За цією ознакою механізми утворюють шість рівнів (табл. 1), упорядкованих за зростанням незалежності джерела істини від досліджуваної моделі та строгості гарантій.

Таблиця 1.

Систематизація механізмів верифікації за джерелом істини

Рівень	**Механізм**	**Зміст перевірки**	**Джерело істини**	**Межа гарантій**
0	Відсутній	Вихід виконується без перевірки	Немає	Гарантій немає
1	Самоперевірка	Та сама модель оцінює власний вихід [1; 2]	Сама модель	Не виявляє хиб, спільних для породження й перевірки; здатна посилювати зміщення [3; 4]
2	Перехресна перевірка	Вихід оцінює інша модель або агент-критик	Інша модель	Лишається ймовірнісною; хиби, спільні для моделей, не виявляються
3	Політика втручання	Правила й монітор допускають, блокують або повертають дію на перегляд [6; 7]	Специфікація політики	Охоплює лише те, що передбачено політикою
4	Виконавча перевірка	Запуск тестів, компіляція, перевірка схеми даних	Середовище виконання	Підтверджує лише перевірені випадки
5	Формальна перевірка	Доведення властивостей виходу до виконання [5]	Формальна специфікація	Потребує специфікації; застосовна до вузького класу властивостей

Практичне значення запропонованого впорядкування полягає в такому. Рівні 1 і 2 гарантій не дають, вони лише знижують імовірність помилки, оскільки джерелом істини лишається ймовірнісна модель. Рівні 3–5 дають гарантії, проте кожен лише в межах свого джерела: політика втручання охоплює передбачені нею випадки, тест підтверджує поведінку на заданих даних, формальне доведення засвідчує відповідність специфікації. Звідси випливає вимога до опису архітектур: зазначати не наявність верифікації, а її рівень за джерелом істини та межі, у яких цей рівень діє. Формулювання «система містить верифікаційний шар» без такого уточнення не несе інформації про гарантії.

Із систематизації випливає обмеження, суттєве для проєктування: зростання рівня підвищує гарантії, проте звужує застосовність. Рівень 4 потребує середовища виконання й набору тестів, а рівень 5 вимагає формальної специфікації бажаної поведінки, яку для багатьох прикладних задач сформулювати важко або економічно недоцільно. Тому рівні комбінують: виконавчу перевірку беруть за основний механізм, політику втручання залучають для дій із незворотними наслідками, а формальну перевірку застосовують вибірково, лише до критичних властивостей на кшталт розмежування доступу до даних.

Висновки. Верифікацію обґрунтовано як самостійний архітектурний шар, що має власний предмет контролю й змінюється незалежно від моделі. Запропоновано систематизацію її механізмів за джерелом істини, яка спирається на класичну проблему оракула, розрізняє шість рівнів і показує, що рівні самоперевірки та перехресної перевірки гарантій не забезпечують, а лише знижують імовірність помилки. Практичним наслідком є вимога зазначати в описі архітектури рівень верифікації та межі його дії, а не сам факт наявності перевірки. Напрямом подальшої роботи є емпіричне зіставлення рівнів за часткою пропущених помилок на спільному наборі задач.

Список літератури

1. Madaan A., Tandon N., Gupta P. et al. Self-Refine: Iterative Refinement with Self-Feedback. arXiv preprint. 2023. DOI: 10.48550/arXiv.2303.17651.

2. Shinn N., Cassano F., Gopinath A. et al. Reflexion: Language Agents with Verbal Reinforcement Learning. Advances in Neural Information Processing Systems 36 (NeurIPS 2023). 2023. DOI: 10.48550/arXiv.2303.11366.

3. Huang J., Chen X., Mishra S. et al. Large Language Models Cannot Self-Correct Reasoning Yet. arXiv preprint. 2023. DOI: 10.48550/arXiv.2310.01798.

4. Xu W., Zhu G., Zhao X. et al. Pride and Prejudice: LLM Amplifies Self-Bias in Self-Refinement. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024. DOI: 10.18653/v1/2024.acl-long.826.

5. Miculicich L., Parmar M., Palangi H. et al. VeriGuard: Enhancing LLM Agent Safety via Verified Code Generation. arXiv preprint. 2025. DOI: 10.48550/arXiv.2510.05156.

6. Hossain E., Nipu M. M. H., Ornee T. N., Rana R., Yousefi N. NEXUS: Structured Runtime Safety for Tool-Using LLM Agents. arXiv preprint. 2026. DOI: 10.48550/arXiv.2607.19356.

7. Bhardwaj V. P. Agent Behavioral Contracts: Formal Specification and Runtime Enforcement for Reliable Autonomous AI Agents. arXiv preprint. 2026. DOI: 10.48550/arXiv.2602.22302.

8. Yu M., Meng F., Zhou X. et al. A Survey on Trustworthy LLM Agents: Threats and Countermeasures. Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2025. DOI: 10.1145/3711896.3736561.

9. Barr E. T., Harman M., McMinn P., Shahbaz M., Yoo S. The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering. 2015. Vol. 41, No. 5. P. 507–525. DOI: 10.1109/TSE.2014.2372785.