AIBuildAI: An AI Agent for Automatically Building AI Models

Ruiyi Zhang [1 *] **Peijia Qin** [1 *] **Qi Cao** [1 *] **Li Zhang** [1 *] **Pengtao Xie** [1]

Abstract

AI models underpin modern intelligent systems, driving advances across science, medicine, and technology. Yet developing high-performing AI models remains a labor-intensive process that requires expert practitioners to iteratively design architectures, engineer representations, implement training pipelines and refine approaches through empirical evaluation. Existing AutoML methods partially alleviate this burden but remain limited to narrow aspects such as hyperparameter optimization and model selection within predefined search spaces, leaving the full development lifecycle largely dependent on human expertise. To address this gap, we introduce AIBuildAI, an AI agent that automatically builds AI models from a task description and training data. AIBuildAI adopts a hierarchical agent architecture in which a manager agent coordinates three specialized sub-agents: a designer for modeling strategy, a coder for implementation and debugging, and a tuner for training and performance optimization. Each sub-agent is itself a large language model based agent capable of multi-step reasoning and tool use, enabling end-to-end automation of the AI model development process that goes beyond the scope of existing AutoML approaches. We evaluate AIBuildAI on MLE-Bench, a benchmark of realistic Kaggle-style AI development tasks. AIBuildAI ranks first on MLE-Bench with a medal rate of 63.1%, outperforming all existing baseline methods and matching the capability of highly experienced AI engineers. These results demonstrate that hierarchical agent systems can automate the full AI model development process from task specification to deployable model, suggesting a pathway toward broadly accessible AI development with minimal human intervention.

[1]Electrical and Computer Engineering, University of California San Diego, CA, US. Correspondence to: Pengtao Xie <p1xie@ucsd.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

AI models underpin modern intelligent systems and are central to progress across biomedicine, chemistry, materials science, the physical sciences, and a broad range of scientific disciplines (Turing, 2007; Jordan & Mitchell, 2015; Ghahramani, 2015; Biamonte et al., 2017). Advances in computer vision, natural language processing, time-series forecasting, tabular data analysis, and related areas have enabled data-driven discovery and decision-making at scale (He et al., 2016; Otter et al., 2020; Price et al., 2025; Hollmann et al., 2025). Despite the availability of mature learning algorithms and widely used deep learning libraries, developing high-performing AI models remains labor-intensive and reliant on specialized expertise (Bergstra & Bengio, 2012; Sculley et al., 2015; Hutter et al., 2019). The process involves iterative design of modeling strategies, data preprocessing and validation, writing code to implement these designs, building and running training pipelines, hyperparameter optimization, debugging, and continual refinement based on empirical evaluation (Aldoseri et al., 2023; Yang et al., 2024). This complexity limits the ability of scientists and practitioners to fully leverage AI for large-scale discovery and real-world applications (Domingos, 2012; Feurer et al., 2015; Henderson et al., 2018).

Existing automation approaches partially alleviate this burden but remain limited in scope. Traditional AutoML methods focus on hyperparameter optimization and model selection within predefined candidate sets (Thornton et al., 2013; He et al., 2021). Despite their effectiveness, AutoML methods require users to have prior knowledge to define candidates and cannot achieve full automation. More recently, large language model (LLM) based coding agents have been applied to automatically building AI models. These methods treat an AI model as a code solution and use a single LLM call to iteratively modify and search for an optimal solution (Toledo et al., 2025; Du et al., 2025). However, these approaches are constrained by what can be accomplished in a single LLM call, which limits the complexity of modifications at each step. Moreover, implementation and tuning of AI models reflect two distinct phases of development with separate objectives, posing additional challenges for a single coding agent to handle.

Recent advances in LLM-based multi-agent systems have

demonstrated that coordinating multiple specialized agents can effectively tackle complex tasks such as software development and scientific research (Hong et al., 2024; Wu et al., 2024). Building on this progress, we introduce AIBuildAI, an AI agent that automatically builds AI models from a task description and training data using multiple coordinated sub-agents. Unlike approaches based on a single coding agent, AIBuildAI adopts a hierarchical agent architecture in which a manager agent coordinates three specialized sub-agents: a designer for modeling strategy, a coder for implementation and debugging, and a tuner for training and performance optimization. Each sub-agent is itself an LLM-based agent that performs multiple rounds of LLM calls and tool interactions within a single invocation; for example, the coder sub-agent may iteratively write code, execute it, inspect errors and refine the implementation across many such rounds before returning a working solution. The manager selects which candidate solutions to develop further through long-context reasoning over the full execution history, replacing predefined search policies with adaptive, context-aware decisions.

We evaluate AIBuildAI on MLE-Bench, a benchmark of realistic Kaggle-style AI development tasks spanning visual, textual, time-series, and tabular modalities (Chan et al., 2025). Without task-specific customization, AIBuildAI ranks first on the March 18, 2026 leaderboard of MLE-Bench (OpenAI, 2026) with a medal rate of 63.1%, outperforming all existing autonomous AI development systems including AIRA-dojo (Toledo et al., 2025) and MLEvolve (Du et al., 2025). These results demonstrate that hierarchical agent systems can automate the full AI model development process, from task specification to deployable model, matching the capability of highly experienced AI engineers with minimal human intervention.

2. Results

2.1. AIBuildAI ranks first on the MLE-Bench

We evaluate AIBuildAI on MLE-Bench, one of the most comprehensive benchmarks for assessing autonomous AI systems on realistic Kaggle-style tasks that require end-to-end AI model development rather than isolated algorithmic components (Chan et al., 2025). MLE-Bench contains 75 tasks curated from past Kaggle competitions, each providing raw datasets, evaluation metrics, and submission protocols faithful to real-world AI development, and requiring systems to design modeling strategies, implement training pipelines, and iteratively improve performance under limited computational budgets. Performance is measured using the medal system derived from human Kaggle competition rankings, where submissions that meet predefined percentile thresholds on the Kaggle leaderboard are awarded medals. As Kaggle competitions draw a global community of expert AI researchers and developers, earning a medal serves as a direct proxy for AI development capability on par with top human practitioners.

AIBuildAI ranks first on the MLE-Bench leaderboard with a medal rate of 63.1% as of March 18, 2026 (Fig. 2a), establishing a new state of the art for autonomous AI development. AIBuildAI outperforms all 26 recent baseline methods (or method variants) on the leaderboard, including MARS (Chen et al., 2026), Famou-Agent (Li et al., 2025), ML-Master (Liu et al., 2025), Leeroo (Nadafian et al., 2026), InternAgent (Team et al., 2025), R&D-Agent (Yang et al., 2025), AIDE (Jiang et al., 2025), AIRA-dojo (Toledo et al., 2025), and MLEvolve (Du et al., 2025). Together, these results demonstrate that AIBuildAI matches the capability of top-tier human AI researchers and developers across a broad spectrum of real-world AI development tasks without any task-specific customization, highlighting the effectiveness of its hierarchical sub-agent design for autonomous AI model development.

Fig. 2b further reports the performance of AIBuildAI across the three complexity splits of MLE-Bench, comprising 22 low-complexity, 38 medium-complexity, and 15 high-complexity tasks, with complexity reflecting the difficulty of building a competitive solution. AIBuildAI achieves medal rates of 77.27%, 61.40%, and 46.67% on the low-, medium-, and high-complexity splits, respectively, ranking first on both the medium- and high-complexity splits. The high-complexity split represents the most challenging and practically important tier of MLE-Bench, demanding multi-step strategy design, careful implementation, and extensive tuning to obtain a competitive solution. The strong performance of AIBuildAI on this tier underscores its advantage in building AI models for the most demanding tasks.

2.2. Performance on image classification tasks

Image classification involves predicting semantic category labels from visual inputs (Krizhevsky et al., 2017; He et al., 2016), commonly addressed by fine-tuning pretrained convolutional neural networks such as ResNet (He et al., 2016) and EfficientNet (Tan & Le, 2019), or vision transformers such as ViT (Dosovitskiy et al., 2021) and Swin Transformer (Liu et al., 2021), potentially with data augmentation strategies (Wang & Deng, 2018; Cubuk et al., 2019). We evaluate AIBuildAI on 10 image classification tasks encompassing natural image recognition (dog-breed-identification), fine-grained species classification (herbarium-2021-fgvc8, herbarium-2022-fgvc9, herbarium-2025-fgvc7, inaturalist-2019-fgvc6, imet-2020-fgvc7), medical imaging classification (brain-tumor-radiogenomic-classification, histopathologic-cancer-detection), data-security-related image classification (alaska2-image-steganalysis), and object recognition (statoil-iceberg-classifier-challenge). As shown in Fig. 3c,

AIBuildAI outperforms both baselines (MLEvolve and AIRA-dojo) on all 10 tasks, demonstrating its strong capability in solving image classification problems across diverse visual domains.

Fig. 3a illustrates an example workflow of AIBuildAI on an image classification task: alaska2-image-steganalysis. It is a steganalysis challenge where the goal is to predict the likelihood that a given image contains a secret message embedded using one of several steganographic algorithms. To address this task, the AIBuildAI manager agent invokes the designer sub-agent to propose six solution candidates based on diverse architectures including EfficientNet (Tan & Le, 2019), ResNet (He et al., 2016), ConvNeXt (Liu et al., 2022), Swin Transformer (Liu et al., 2021), and Wide-ResNet (Zagoruyko & Komodakis, 2016) variants. The manager reviews the six plans and selects four to proceed, invoking the coder sub-agent to implement each plan and the tuner sub-agent to optimize training parameters and obtain validation results. Based on these results, the manager invokes the designer sub-agent to revise candidate 2 by removing stride surgery and reverting to vanilla EfficientNet-B4, which then passes through the coder and tuner again. Finally, the aggregator agent constructs a rank-based ensemble of the four models, achieving a score of 0.8101 on the private test set.

Fig. 3b depicts the final solution of AIBuildAI on the alaska2-image-steganalysis task in detail. The pipeline takes an image as input, resizes and normalizes it using ImageNet (Deng et al., 2009) statistics. Test-time augmentation is applied via horizontal flips, vertical flips, and the combination of both, producing four views per image (original, H-flip, V-flip, H+V-flip). Each view is independently passed through the four ensemble models, namely EfficientNet-V2-S (Tan & Le, 2021) with focal loss (Lin et al., 2017), EfficientNet-B4 (Tan & Le, 2019), ResNet50 (He et al., 2016) with a learnable high-pass filter bank, and Wide-ResNet-50-2 (Zagoruyko & Komodakis, 2016) with a dual-branch pixel-and-edge architecture. Each model produces per-image raw scores, which are then converted to rank percentiles normalized to $[0, 1]$ across the test set. The rank percentiles from the four models are combined via weighted averaging to produce a final stego score between 0 and 1, where higher values indicate a greater likelihood of steganographic content.

2.3. Performance on other vision tasks

Detection, segmentation, and video understanding involve predicting spatially or temporally structured outputs from visual inputs (Liu et al., 2020; Sharma et al., 2022), typically addressed using architectures such as Faster R-CNN (Ren et al., 2015) and YOLO (Redmon et al., 2016) for object detection, UNet (Ronneberger et al., 2015) and DeepLabV3+ (Chen et al., 2017) for semantic segmentation, and two-stream networks (Simonyan & Zisserman, 2014) or video transformers (Arnab et al., 2021) for temporal modeling. We evaluate AIBuildAI on 10 visual perception tasks involving structured prediction from images or videos, including object detection (3d-object-detection), semantic segmentation (contrails-identify, tgs-salt-identification-challenge), medical imaging (rsna-cervical-spine-fracture-detection, siim-covid19-detection, chest-xray-abnormalities-detection, pulmonary-fibrosis-progression), character recognition (kuzushiji-recognition), pet scoring regression (petfinder-pawpularity-score), and video-based activity recognition (nfl-player-contact-detection). As shown in Fig. 4c, AIBuildAI outperforms both baselines (MLEvolve and AIRA-dojo) on all 10 tasks, highlighting its effectiveness in handling spatially and temporally structured visual prediction tasks.

Fig. 4a illustrates an example workflow of AIBuildAI on a semantic segmentation task: contrails-identify. It aims to generate semantic masks of contrails (condensation trails formed by aircraft engines) from satellite images. To address this task, the AIBuildAI manager agent invokes the designer sub-agent to propose seven solution candidates based on diverse segmentation architectures including UNet (Ronneberger et al., 2015), FPN-ResNet (He et al., 2016), DeepLabV3+ (Chen et al., 2017), and ConvNeXt (Liu et al., 2022) variants. The manager reviews the seven plans, selects a subset of four to proceed, and invokes the coder and tuner sub-agents to implement and evaluate them. Based on the initial results, the manager invokes the designer sub-agent to revise candidates 1 and 5: for candidate 1, it adds a Lovász-Dice loss (Berman et al., 2018; Milletari et al., 2016) component and implements test-time augmentation; for candidate 5, it introduces a cosine learning rate scheduler for the optimizer. The coder sub-agent implements the revised designs and the tuner sub-agent retrains the updated models. Finally, the aggregator agent constructs a weighted ensemble of the two strongest candidates, achieving a Dice score of 0.660 on the private test set.

Fig. 4b depicts the final solution of AIBuildAI on the contrails-identify task in detail. The pipeline takes a satellite image as input and feeds it through two parallel branches. The first branch uses a UNet (Ronneberger et al., 2015) with an EfficientNet-B4 (Tan & Le, 2019) encoder. It selects specific satellite spectral bands and combines them into a false-color image that highlights contrails, applies per-image Z-score normalization, and upsamples the input to a higher resolution for segmentation. The second branch uses a DeepLabV3+ (Chen et al., 2017) with a SegFormer encoder (MiT-B3) (Xie et al., 2021) fine-tuned on ImageNet-1k (Deng et al., 2009), using a different set of satellite bands with global standardization. Each model produces a per-

pixel probability map indicating contrail presence, which is upsampled back to the original input resolution. The two probability maps are then combined with a 0.3/0.7 weighting (UNet/DeepLabV3+) and thresholded at 0.8 to produce binary contrail masks.

2.4. Performance on language tasks

Language understanding and generation require modeling textual data for both classification and sequence prediction objectives (Bengio et al., 2003), with standard approaches relying on fine-tuning pretrained language models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) for classification and extraction tasks, and sequence-to-sequence models such as T5 (Raffel et al., 2020) and GPT (Radford et al., 2019) for generation tasks. We evaluate AIBuildAI on 10 language-related tasks, covering text classification (patent-matching, lmsys-chatbot-arena, spooky-author-identification), question answering (chaii-hindi-and-tamil-question-answering, google-quest-challenge), information extraction (facebook-recruiting-iii-keyword-extraction), sequence-to-sequence generation (text-normalization-challenge-english-language), word imputation (billion-word-imputation), sentiment analysis (tweet-sentiment-extraction), and text-based prediction (random-acts-of-pizza). As shown in Fig. 5c, AIBuildAI outperforms both baselines (MLEvolve and AIRA-dojo) on all 10 tasks, demonstrating its effectiveness in solving diverse language understanding and generation problems.

Fig. 5a illustrates an example workflow of AIBuildAI on a language understanding task: billion-word-imputation. It aims to predict a missing word at an unknown position in a given text paragraph. To address this task, the AIBuildAI manager agent invokes the designer sub-agent to propose seven solution candidates based on diverse approaches including BERT-base MLM (Devlin et al., 2019), BERT-large, DistilBERT (Sanh et al., 2019), n-gram statistical methods, and multi-model ensembles. The manager reviews the seven plans, selects a subset of four to proceed, and invokes the coder and tuner sub-agents to implement, train and evaluate them. Based on the preliminary results, the manager invokes the designer sub-agent to revise candidate 1 by incorporating a token-level masking method with word-to-token mapping. The coder sub-agent implements the revised design and the tuner sub-agent retrains the model. Finally, the aggregator agent evaluates the completed submissions and selects candidate 1 based on validation metrics, achieving a Levenshtein distance of 5.519 on the private test set.

Fig. 5b depicts the final solution of AIBuildAI on the billion-word-imputation task in detail. Given an input sentence with a missing word at an unknown position, the pipeline generates candidate sentences by inserting a mask token at every possible position. Each candidate is tokenized using a wordpiece tokenizer and fed into a BERT-base-uncased model (Devlin et al., 2019), which computes the log-probability over the vocabulary at the masked position and extracts the score of the most likely predicted word. The position scores across all candidates are compared, and the position with the highest confidence is selected. A confidence gate then determines whether the prediction is reliable: if the confidence exceeds a threshold, the mask is replaced with the predicted word to produce the reconstructed sentence; otherwise, the original input is returned.

3. Discussion

AIBuildAI achieves superior performance by employing a specialized sub-agent architecture that decomposes the complex model development process into distinct, iterative roles like designing, coding, and tuning. Unlike earlier systems such as AIRA and MLEvolve, which rely on a single LLM call to transform code repositories and are thus limited by what can be accomplished in one shot, AIBuildAI's sub-agents use multi-step reasoning and tool interactions to refine solutions in parallel. This method effectively prevents "context explosion" by ensuring each agent only manages information relevant to its specific task; for instance, a coder can independently debug and rewrite scripts through multiple execution trials without overwhelming the tuner, which only requires the final, working code to begin training optimization. Although this sophisticated interaction naturally increases token consumption and communication overhead, the system could become more economical through dynamic routing, where simpler tasks like configuration updates are handled by smaller models while high-level reasoning is reserved for the most capable ones. To further its impact, future development aims to scale AIBuildAI beyond limited single-GPU setups to handle massive, production-grade computational resources—including multi-GPU training and distributed parallelism—and to integrate a structured knowledge base of proven AI patterns to guide decisions. Furthermore, incorporating a retrievable collection of modeling strategies and preprocessing techniques from literature would allow the system to accumulate experience and improve its consistency across different tasks over time. Ultimately, by automating the entire pipeline from initial strategy design to final model optimization, AIBuildAI has the potential to transform data-rich fields like genomics, materials science, and drug discovery, where data is abundant but AI expertise is scarce. This capability allows scientists to rapidly build high-performing models for tasks such as cell-type annotation or virtual screening, ensuring that the ability to transform raw data into deployable AI becomes as transformative for discovery as the high-throughput instruments that generate the data themselves.

References

Abolhasani, M. and Kumari, E. The rise of self-driving laboratories in chemical and materials sciences. *Nature Synthesis*, 2:483–492, 2023.

Acharya, J. N., Hani, A. J., Thirumala, P. D., and Tsuchida, T. N. American clinical neurophysiology society guideline 3: A proposal for standard montages to be used in clinical EEG. *Journal of Clinical Neurophysiology*, 33(4): 312–316, 2016. doi: 10.1097/WNP.0000000000000317.

Aldoseri, A., Al-Khalifa, K. N., and Hamouda, A. M. Rethinking data strategy and integration for artificial intelligence: concepts, opportunities, and challenges. *Applied Sciences*, 13(12):7082, 2023.

Anthropic. Claude opus 4.6 model card. https://docs.anthropic.com/en/docs/about-claude/models, 2026. Accessed: 2026-03-18.

Arik, S. Ö. and Pfister, T. TabNet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:6679–6687, 2021.

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. ViViT: A video vision transformer. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6836–6846, 2021.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, abs/1803.01271, 2018. URL https://api.semanticscholar.org/CorpusID:4747877.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Berman, M., Triki, A. R., and Blaschko, M. B. The Lovász-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. pp. 4413–4421, 2018.

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

Butterworth, S. On the theory of filter amplifiers. *Experimental Wireless and the Wireless Engineer*, 7:536–541, 1930.

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Madry, A., and Weng, L. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. International Conference on Learning Representations (ICLR).

Chen, J., Dalvi Mishra, B., Nam, J., Meng, R., Pfister, T., and Yoon, J. MARS: Modular agent with reflective search for automated AI research. *arXiv preprint arXiv:2602.02660*, 2026.

Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. Rethinking atrous convolution for semantic image segmentation, 2017.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system, 2016. URL https://doi.org/10.1145/2939672.2939785.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation strategies from data., 2019. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2019.html#CubukZMVL19.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL), 2019.

Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Ruhle, V., Lakshmanan, L. V., and Awadallah, A. H. Hybrid LLM: Cost-efficient and quality-aware query routing. *Proceedings of the Twelfth International Conference on Learning Representations*, 2024.

Domingos, P. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Du, S., Yan, X., Jiang, D., Yuan, J., Hu, Y., Li, X., He, L., Zhang, B., and Bai, L. Automlgen: Navigating fine-grained optimization for coding agents. *ArXiv*, abs/2510.08511, 2025. URL https:

//api.semanticscholar.org/CorpusID:281951479.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 2755–2763, 2015.

Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

Google. Gemini 3 model card. https://ai.google.dev/gemini-api/docs/models, 2025. Accessed: 2025-12-11.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data?, 2022. URL https://openreview.net/forum?id=Fp7__phQszn.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Agapito, L., Berg, T., Kosecka, J., and Zelnik-Manor, L. (eds.), *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

He, X., Zhao, K., and Chu, X. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., and Wilson, K. W. Cnn architectures for large-scale audio classification. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 131–135, 2016. URL https://api.semanticscholar.org/CorpusID:8810481.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated machine learning: methods, systems, challenges*. Springer, 2019.

Jasper, H. H. The ten-twenty electrode system of the International Federation. *Electroencephalography and Clinical Neurophysiology*, 10:371–375, 1958.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. AIDE: AI-Driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jordan, M. I. and Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: a highly efficient gradient boosting decision tree, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

Li, A., Wu, C., Ge, Z., Chong, Y. H., Hou, Z., Cao, L., Ju, C., Wu, J., Li, H., Zhang, H., Feng, S., Zhao, M., Qiu, F., Yang, R., Zhang, M., Zhu, W., Sun, Y., Sun, Q., Yan, S., Liu, D., Yin, D., and Shen, D. The FM agent. *arXiv preprint arXiv:2510.26144*, 2025.

Lin, T.-Y., Goyal, P., Girshick, R. B., He, K., and Dollár, P. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327, 2017. URL https://api.semanticscholar.org/CorpusID:206771220.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. Deep learning for generic object detection: A survey. *Int. J. Comput. Vision*, 128 (2):261–318, February 2020. ISSN 0920-5691. doi: 10.1007/s11263-019-01247-4. URL https://doi.org/10.1007/s11263-019-01247-4.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. arXiv, 2019.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

Liu, Z., Cai, Y., Zhu, X., Zheng, Y., Chen, R., Wen, Y., Wang, Y., E, W., and Chen, S. ML-Master: Towards AI-for-AI via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*, 2025.

Martí, R., Resende, M. G. C., and Ribeiro, C. C. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226(1):1–8, 2013.

Milletari, F., Navab, N., and Ahmadi, S.-A. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. *Proceedings of the International Conference on 3D Vision (3DV)*, pp. 565–571, 2016.

Nadafian, A., Mohammadshahi, A., and Yazdani, M. KAPSO: A knowledge-grounded framework for autonomous program synthesis and optimization. *arXiv preprint arXiv:2601.21526*, 2026.

Novikov, A., V˜u, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B. M., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., Balog, M., and DeepMind, G. Alphaevolve: A coding agent for scientific and algorithmic discovery. *ArXiv*, abs/2506.13131, 2025. URL https://api.semanticscholar.org/CorpusID:278658695.

OpenAI. Introducing OpenAI o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025. Accessed: 2025-04-16.

OpenAI. Mle-bench leaderboard (commit c5631ba). https://github.com/openai/mle-bench/tree/c5631ba61ceeb0573235a6ce209db435327a1e84, 2026. Accessed: 2026-03-18.

Otter, D. W., Medina, J. R., and Kalita, J. K. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

Peidli, S., Green, T. D., Shen, C., Gross, T., Min, J., Garda, S., Yuan, B., Schumacher, L. J., Taylor-King, J. P., Marks, D. S., Luna, A., Schaar, M. v. d., and Theis, F. J. scPerturb: Harmonized single-cell perturbation data. *Nature Methods*, 21:531–540, 2024.

Price, I., Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., Ewalds, T., Stott, J., Mohamed, S., Battaglia, P., et al. Probabilistic weather forecasting with machine learning. *Nature*, 637(8044):84–90, 2025.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. OpenAI Technical Report, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), January 2020. ISSN 1532-4435.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

Ren, S., He, K., Girshick, R., and Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28, 2015.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 2023. doi: 10.1038/s41586-023-06924-6.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015.

Sadybekov, A. V. and Katritch, V. Computational approaches streamlining drug discovery. *Nature*, 616(7958): 673–685, 2023. doi: 10.1038/s41586-023-05905-z.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv, 2019.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. Hidden technical debt in machine learning systems. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Proceedings of the International Conference on Neural Information Processing Systems*, volume 2, pp. 2503–2511, 2015.

Sharma, R., Saqib, M., Lin, C. T., and Blumenstein, M. A survey on object instance segmentation. *SN Comput. Sci.*, 3(6), September 2022. doi: 10.1007/s42979-022-01407-3. URL https://doi.org/10.1007/s42979-022-01407-3.

Simonyan, K. and Zisserman, A. Two-stream convolutional networks for action recognition in videos. Advances in Neural Information Processing Systems (NeurIPS), 2014.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

Tan, M. and Le, Q. V. EfficientNetV2: Smaller models and faster training. *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 10096–10106, 2021.

Team, I., Zhang, B., Feng, S., Yan, X., Yuan, J., Ma, R., Hu, Y., Yu, Z., He, X., Huang, S., et al. InternAgent: When agent becomes the scientist—building closed-loop system from hypothesis to verification. *arXiv preprint arXiv:2505.16938*, 2025.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In Dhillon, I. S., Koren, Y., Ghani, R., Senator, T. E., and Schmerl, B. (eds.), *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855, 2013.

Toledo, E., Hambardzumyan, K., Josifoski, M., HAZRA, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., Lupu, A., Raileanu, R., Shavrina, T., Niu, K., Gagnon-Audet, J.-C., Shvartsman, M., Sodhani, S., Miller, A. H., Charnalia, A., Dunfield, D., Wu, C.-J., Stenetorp, P., Cancedda, N., Foerster, J. N., and Bachrach, Y. AI research agents for machine learning: Search, exploration, and generalization in MLE-bench. 2025. URL https://openreview.net/forum?id=RwfrdKSgCE.

Turing, A. M. *Computing machinery and intelligence*, pp. 23–65. Springer, 2007.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Wang, M. and Deng, W. Deep visual domain adaptation: A survey. *Neurocomput.*, 312(C):135–153, October 2018. ISSN 0925-2312. doi: 10.1016/j.neucom.2018.05.083. URL https://doi.org/10.1016/j.neucom.2018.05.083.

Wang, X., Liu, Y., Cheng, W., Zhao, X., Chen, Z., Yu, W., Fu, Y., and Chen, H. MixLLM: Dynamic routing in mixed large language models. *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025.

Wang, Z., Li, Z., Jiang, Z., Tu, D., and Shi, W. Crafting personalized agents through retrieval-augmented generation on editable memory graphs. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Wu, K., Zhang, J., Peng, H., Liu, M., Xiao, B., Fu, J., and Yuan, L. TinyViT: Fast pretraining distillation for small vision transformers. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *Conference on Language Modeling (COLM)*, 2024.

Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., and Luo, P. SegFormer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021.

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-MEM: Agentic memory for LLM agents. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yang, X., Yang, X., Fang, S., Zhang, Y., Wang, J., Xian, B., Li, Q., Li, J., Xu, M., Li, Y., Pan, H., Zhang, Y., Liu, W., Shen, Y., Chen, W., and Bian, J. R&D-Agent: An LLM-Agent framework towards autonomous data science. *arXiv preprint arXiv:2505.14738*, 2025.

Yang, Y., Zhang, H., Gichoya, J. W., Katabi, D., and Ghassemi, M. The limits of fair medical imaging ai in real-world generalization. *Nature medicine*, 30(10):2838–2848, 2024.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E. P., Gonzalez, J. E., and Stoica, I. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

Zhu, Z., Giannoula, C., Andoorveedu, M., Su, Q., Mangalam, K., Zheng, B., and Pekhimenko, G. Mist: Efficient distributed training of large language models via memory-parallelism co-optimization. *Proceedings of the 20th European Conference on Computer Systems (EuroSys)*, 2025.

A. Method details

A.1. Overview of AIBuildAI

AIBuildAI is an AI agent for automatically building AI models. It takes as input a text description of an AI task and training data, automatically designs, implements and trains an AI model with high performance, and outputs the trained model checkpoints and inference script that can be directly applied to unseen test data. AIBuildAI adopts a hierarchical agent architecture inspired by real-world AI research processes. In traditional AI projects, a technical lead oversees several parallel lines of investigation, each exploring a different modeling direction. Within each line, researchers design modeling strategies while engineers implement training pipelines and run experiments to obtain empirical results. The lead periodically reviews the outcomes of these experiments, allocates resources to promising directions, requests revisions when progress stalls, and terminates unproductive efforts to focus on stronger candidates. AIBuildAI mirrors this process through a structured, agent-driven workflow (Fig. 1a).

The core workflow of AIBuildAI operates over multiple parallel solution candidates, each maintained as an isolated workspace (repository) containing the artifacts that define an AI modeling pipeline, including the model design description, training code, configuration and hyperparameter files, logs, and evaluation results. Solution repositories are initialized empty and iteratively refined until the time budget is exhausted, under a trio of core sub-agents (designer, coder, and tuner) orchestrated by a manager agent. Each sub-agent is capable of multi-step reasoning and tool usage through multiple LLM calls within a single agent invocation. The designer sub-agent is responsible for modeling and training strategy, proposing new approaches informed by web search or revising existing plans based on training and evaluation outcomes. The coder sub-agent focuses on producing correct code from a textual design description, with operations that include reading the design, translating it into executable training and inference pipelines, writing and running code, inspecting errors, refining the implementation, and performing iterative debugging. The tuner sub-agent trains the model and tunes parameters to maximize performance, with operations that include launching training runs, monitoring metrics such as losses, reasoning over logs, adjusting hyperparameters, and iterating. A manager agent orchestrates this process by monitoring the current status across solution repositories and deciding which sub-agent to invoke on each. This requires long-context reasoning over the accumulated execution history, adaptively composing different sequences of operations depending on the needs of each solution. In principle, promising solutions are further refined through targeted updates, while underperforming ones are discarded by ceasing further actions. Surrounding this core loop are two lightweight components: a setup agent that sets up the conda environment with the necessary packages, and an aggregator agent that selects or ensembles the strongest candidates to produce the final submission-ready model.

AIBuildAI is applicable to a diverse range of AI tasks across multiple data modalities and problem formulations, including visual recognition, natural language processing, temporal forecasting, and structured tabular prediction (Fig. 1b). In the following sections, we first evaluate AIBuildAI on MLE-Bench against state-of-the-art baseline methods, and then compare its detailed task performance against two representative baselines, AIRA-dojo (Toledo et al., 2025) and MLEvolve (Du et al., 2025), across 10 image classification tasks, 10 detection, segmentation, and video understanding tasks, 10 language understanding and generation tasks, and 6 temporal and tabular tasks.

A.2. LLM-based agent abstraction

A large language model (LLM) generates outputs conditioned on an input prompt. In a standard single-shot usage, the model is invoked once and its output is returned as the final result. An LLM-based agent extends this paradigm by allowing the model to iteratively interact with an external environment through tool use. Instead of producing only a final answer, the model may generate intermediate outputs that request the execution of external tools (e.g., running code, querying a database, or modifying files). The results of these tool executions are then appended to the context, enabling the model to produce the final answer based on additional information from tool calls.

Formally, an agent is defined by an agent-specific prompt p_a and a set of available tools. Given a user input u, the agent first invokes the language model to produce an initial output $y_1 = p_\theta(p_a, u)$. A control loop then examines y_1: if it contains a tool invocation, the environment executes the requested tool and returns the tool execution result (observation) o_1; otherwise the agent terminates and returns y_1 as the final output. If there is a tool invocation, the observation is appended to the context of the agent and the model is invoked again to produce $y_2 = p_\theta(p_a, u, y_1, o_1)$. In general, at step t the model generates $y_t = p_\theta(p_a, u, y_1, o_1, \ldots, y_{t-1}, o_{t-1})$, conditioned on the prompt and the full interaction history. Similarly, if y_t contains a tool invocation, the resulting observation o_t is appended to the history. The loop continues until the model produces a response without a tool invocation, or a maximum number of steps is reached. Under this abstraction, a specific agent is



Figure 1. **Overview of AIBuildAI. a**, AIBuildAI is an AI agent for automatically building AI models. It takes as input a text description of an AI task and training data, automatically designs and trains an AI model, and outputs the trained model checkpoints and inference script. AIBuildAI adopts a hierarchical agent architecture in which a manager agent oversees multiple parallel solutions, deciding which sub-agent to invoke for iteratively refining each solution repository. Sub-agents include a designer that proposes or revises modeling plans, a coder that implements and debugs training pipelines, and a tuner that runs full training and optimizes performance. Each sub-agent is itself an LLM-based agent that performs multiple rounds of LLM calls and tool interactions within a single invocation. Surrounding this core loop, a setup agent prepares the conda environment and installs the required packages at initialization, and an aggregator agent selects or ensembles the strongest candidate solutions at termination to produce the final AI model. **b**, AIBuildAI is applicable to a diverse range of tasks spanning multiple data modalities and problem formulations, including visual recognition, natural language processing, temporal forecasting, and structured tabular prediction.

fully specified by (i) its prompt p_a, which describes the task and how the model should act when intermediate results are presented to the model, and (ii) its tool set, which defines the tools the agent can invoke. By varying these two components with the same underlying language model, one can construct agents with different capabilities and execution strategies.

A.3. AIBuildAI system design

We consider the task of automated AI model development, where the goal is to automatically construct a complete solution for a given AI task. The input consists of (i) a natural language description of the AI task and its evaluation protocol, and (ii) a dataset. The desired output is a runnable solution repository, including trained model checkpoints and an inference script that produces predictions for the task on unseen data.

Our approach shares the high-level idea of multi-start local search (Martí et al., 2013). We initialize n solution repositories, denoted $\{\text{repo}_i\}_{i=1}^n$, each representing a fundamentally different modeling strategy for solving the task. Each repository provides an isolated workspace initialized with a copy of the input data, within which a candidate solution can be independently developed and trained. Once initialized, the solution within each repository is iteratively refined: analogous to local search, improvements are incremental rather than fundamental redesigns. At termination, the best-performing solutions are selected or ensembled to produce the final output. Each repository repo_i maintains a structured collection of artifacts: data_i denotes a copy of the training data, plan_i denotes the solution strategy in text, code_i denotes the implementation of the model and training loop, config_i contains hyperparameters and training configurations, and result_i contains execution outputs including training loss, validation performance, and model checkpoints.

To carry out this strategy, we instantiate AIBuildAI as an LLM-based agent following the abstraction described above, with repository-scoped tool variants read_i, write_i, and execute_i for file manipulation and program execution within each repo_i, as well as a search tool for retrieving external references on the Internet. The agent iteratively refines each solution trajectory by reading artifacts, modifying plans, code, or configurations, and executing training runs. In principle, a single agent with access to all tools across all repositories could carry out this process. However, refining n parallel solutions involves heterogeneous operations (designing strategies, writing and debugging code, adjusting hyperparameters) across independent repositories, and maintaining all of this information in a single context would rapidly exceed practical context limits and dilute the agent's attention with unrelated details from different methods and repositories.

To address this, AIBuildAI adopts a hierarchical design in which the primary agent (manager) delegates complex operations to specialized sub-agents. Concretely, sub-agents are exposed as tools in the manager's tool set and are invoked by the manager to perform a specific type of refinement on a candidate solution. AIBuildAI includes sub-agents for solution design (designer sub-agent), code generation (coder sub-agent), and model training and hyperparameter tuning (tuner sub-agent), which are detailed in the following subsections. By delegating detailed reasoning and execution to these sub-agents, the manager operates with a compact context that is essential for effectively maintaining a unified control flow across an entire AIBuildAI run. The sub-agents are themselves implemented as LLM-based agents with their own prompts and restricted subsets of the overall tool interface. When invoked, a sub-agent maintains its own internal context with sequences of LLM calls and tool execution results, and writes the resulting artifacts to its repository. This design enables decomposition of the overall task into modular components, enabling more focused and effective behavior.

A.4. Manager

The manager is a single top-level agent that orchestrates the overall solution process across all n repositories by selecting and invoking tools based on the current context. Its tool set consists of repository-scoped sub-agent tools (designer_i, coder_i, tuner_i) and read_i tools for i from 1 to n. The sub-agent tools invoke specialized agents that operate on each repository. The manager understands each sub-agent's behavior through its tool description, which specifies that the designer sub-agent generates or revises solution strategies, the coder sub-agent implements or modifies solutions, and the tuner sub-agent executes training and improves model performance, enabling the manager to reason about when to invoke each. The read_i tools provide the manager with access to artifacts across all repositories, including plans, code, and results. This enables comparing candidate solutions to select promising ones to proceed with, and understanding the current status of a candidate solution to assign the appropriate sub-agent to refine it. Tool invocations are asynchronous: the manager may issue multiple tool calls in parallel across repositories, allowing concurrent exploration and overlapping execution of design, implementation, and training.

The behavior of the manager is defined by its prompt, which specifies the AI task, the available tools, and the structure of the repository space, and thereby determines how actions are selected during execution. At each step, the manager observes

artifacts (via read_i) and selects actions on specific repositories. In typical executions, it initializes multiple repositories by invoking designer_i to propose candidate strategies, followed by coder_i and tuner_i to implement and evaluate them. Subsequent decisions are made adaptively based on repository-specific results. For example, if a solution underperforms due to flaws in the overall approach, the manager may invoke the designer sub-agent to revise the strategy; if implementation issues are detected, it may call the coder sub-agent to modify the code; if performance is limited by training configuration, it may repeatedly invoke the tuner sub-agent to refine hyperparameters. Promising repositories are further refined through such targeted updates, while underperforming ones or similar ones are discarded by ceasing further actions. This flexible, feedback-driven control contrasts with fixed workflows, enabling the manager to dynamically compose different sequences of sub-agent invocations depending on the needs of each candidate solution.

A.5. Designer

For each of the n repositories, a separate designer sub-agent is instantiated. Here we describe the i-th designer, which is invoked by the manager through the designer_i tool to operate on repo_i. Its tool set contains read_i for accessing artifacts in repo_i, write_i for creating or modifying files, and search for retrieving external references and best practices. At each invocation, the designer observes the current repository contents via read_i and produces an updated plan plan_i, adapting its behavior based on the state of the repository. If plan_i is absent, it generates an initial strategy plan_i that specifies the modeling approach, data processing pipeline, and training procedure, usually with the help of the web search tool. If plan_i and result_i are available, it performs a revision step by analyzing result_i (e.g., training loss, validation metrics, and logs) to identify potential failure modes such as underfitting, overfitting, or optimization instability, and updates plan_i accordingly, for example by modifying model architectures, adjusting data processing pipelines, or changing training procedures. The updated plan is written back to repo_i and subsequently consumed by the coder sub-agent for implementation.

A.6. Coder

For each of the n repositories, a separate coder sub-agent is responsible for implementing a solution plan. Here we describe the i-th coder, which is invoked by the manager through the coder_i tool to operate on repo_i. Its tool set contains read_i for retrieving artifacts, write_i for creating or modifying files, and execute_i for executing Python programs in the repository workspace. The coder has no access to other repositories. At each invocation, the coder observes the current repository contents via read_i, in particular plan_i, and produces or updates the implementation code_i, adapting its behavior based on the state of the repository. If code_i is absent, it generates an initial implementation from plan_i, including the model, training and inference code, and an example configuration file config_i. If code_i already exists, it performs a revision step to fix errors in code_i, improve robustness, or incorporate updates to plan_i. After implementation, the coder invokes execute_i to perform a short verification run, ensuring that the pipeline executes end-to-end correctly. If errors are encountered, it conducts iterative debugging until the code is bug-free. The resulting correct code is written to repo_i for subsequent use by other sub-agents, especially the tuner.

A.7. Tuner

For each of the n repositories, a separate tuner sub-agent is responsible for training models and improving performance. Here we describe the i-th tuner, which is invoked by the manager through the tuner_i tool to operate on repo_i. Its tool set contains read_i for retrieving artifacts, write_i for modifying configuration files config_i, and execute_i for executing training runs in the repository workspace. The tuner has no access to other repositories. At each invocation, the tuner observes the current repository contents via read_i, in particular code_i and result_i, and produces updated results result_i by changing configuration files based on prior execution outcomes and running the code. Given existing result_i (including training logs, validation metrics, and checkpoints), it analyzes learning behavior to identify issues such as slow convergence, overfitting, or instability. It then updates training configurations (e.g., hyperparameters, schedules, or data-related settings) and launches new runs via execute_i. As the manager assigns the tuner a maximum tuning time, it uses short preliminary runs to obtain early performance signals. If performance is stagnant or degrades, ineffective configurations are replaced early. In later stages, the tuner extends promising configurations to full training to achieve optimal results within the time budget. All updated configurations and result_i (logs, checkpoints, and evaluation outputs) are written back to repo_i, delivering validation results to other agents.

A.8. Setup and aggregation

In addition to the manager-controlled workflow, AIBuildAI includes two auxiliary sub-agents that are each invoked once: a setup agent at initialization and an aggregator at termination. These agents are not part of the iterative decision loop, but provide deterministic preparation and finalization of the complete workflow.

Setup The setup agent initializes the model training environment before execution begins. Given the task description, it creates a conda environment and installs the packages required by the subsequent model training process, using conda or pip as appropriate. This ensures that all dependencies are available before any sub-agent begins execution.

Aggregator The aggregator finalizes the run by producing a single submission from the repositories generated during execution. It operates in a dedicated repository repo_{n+1}, separate from all candidate repositories, to avoid modifying any intermediate artifacts. Its tool set contains read_i for accessing artifacts across all candidate repositories, and write_{n+1} and execute_{n+1} for producing the final submission artifact and executing ensemble inference scripts within repo_{n+1}. The behavior of the aggregator is defined by its prompt p_{agg}, which specifies the objective of maximizing final performance by combining or selecting candidate solutions.

At invocation, the aggregator inspects $\{\text{result}_i\}_{i=1}^{n}$ and determines whether to select a single best candidate or construct a weighted ensemble, based on the diversity of candidate solutions, their individual validation performance, and the remaining computational budget. When candidates are sufficiently diverse and complementary, the aggregator constructs an ensemble by assigning combination weights derived from validation scores. The specific aggregation strategy is adapted to the task: for classification and detection tasks, the aggregator may apply rank-based blending, in which per-model predictions are converted to rank percentiles before weighted averaging; for segmentation tasks, it may combine predicted probability maps with task-tuned weights and apply binary thresholding; for regression or ranking tasks, it may compute weighted averages of raw predictions. When the remaining time budget is insufficient to execute ensemble inference, or when a single candidate substantially outperforms all others, the aggregator instead selects the best-performing repository and directly retrieves its prediction outputs. In all cases, the aggregator does not retrain models or modify intermediate artifacts, ensuring that the final result is deterministically derived from the executed workflow.

A.9. Baseline methods

We compare AIBuildAI with representative approaches for autonomous machine-learning engineering that formulate the problem as a search over code-based solutions. These methods are conceptually related to recent LLM-driven program optimization frameworks (Romera-Paredes et al., 2023; Novikov et al., 2025), which iteratively generate, evaluate, and refine candidate programs. Under this formulation, each candidate solution is a code artifact code that implements a complete training and inference pipeline, and is evaluated by executing it to obtain $\text{result} = \text{execute}(\text{code})$ (e.g., validation performance). The search process constructs a tree over candidate solutions: starting from an initial solution, the system selects a previously evaluated solution code_t (a parent node) and expands it into multiple new variants (child nodes) by modifying code_t conditioned on result_t. Each new variant is then evaluated by executing its training pipeline, which is computationally expensive. To efficiently allocate computation, these methods employ search strategies, often Monte Carlo Tree Search (MCTS), to balance exploration of new variants and exploitation of high-performing candidates.

AIRA AIRA (Toledo et al., 2025) instantiates the expansion step through three core operators, each corresponding to a single LLM call that modifies code_t conditioned on result_t. A Draft operator generates an initial code_t when no previous solution exists; if result_t contains an execution error, a Debug operator revises code_t to produce a corrected variant; if result_t is valid but performance is unsatisfying, an Improve operator adjusts hyperparameters, model architectures, or feature engineering in code_t. A global Memory operator maintains a running summary of all previous solutions and their results, which is appended as context to every Draft and Improve call. The default search policy is greedy: the system first invokes Draft n_d times to seed an initial population, then at each subsequent step selects the highest-scoring code_t and applies Improve if it is valid, or Debug otherwise. AIRA also explores alternative search policies, including MCTS and evolutionary methods, though its analysis emphasizes that system effectiveness is primarily determined by the operator set rather than the choice of search algorithm.

MLEvolve MLEvolve (Du et al., 2025) shares the same core operators as AIRA (Draft, Debug, Improve), each implemented as a single LLM call, but differs in three key aspects. First, it replaces the tree-structured search with Monte Carlo

Graph Search (MCGS) by introducing a multi-branch aggregation operator that combines top trajectories from multiple branches to seed entirely new solutions from the root, forming a graph structure that enables information flow beyond standard parent-child relationships while keeping selection and backpropagation on the tree backbone. Second, it removes the global Memory operator and instead introduces two reference-based operators: an intra-branch evolution operator that supplies the nearest k ancestor nodes from the same branch as context, enabling the LLM to reflect on past successes and failures within a single line of investigation; and a cross-branch reference operator that provides top-performing nodes from other branches as context when a branch stagnates, allowing the LLM to incorporate strategies discovered elsewhere. Third, MLEvolve incorporates a domain-specific knowledge base, curated from open-source repositories and competition platforms, organized along model-level (architecture selection), data-level (preprocessing and feature engineering), and strategy-level (test-time augmentation, ensembling) dimensions. Model-level knowledge is optionally injected during initial Draft generation, while data- and strategy-level knowledge is provided as context during subsequent expansion calls.

A.10. Experimental settings

All experiments follow the MLE-Bench evaluation protocol (Chan et al., 2025). AIBuildAI uses Claude Opus 4.6 (Anthropic, 2026) as the backbone LLM for all agents, with a sampling temperature of 1.0 and no nucleus or top-k truncation, and initializes $n = 7$ parallel solution repositories per task. It is evaluated with a budget of 24 hours on a Linux x86-64 machine with 24 vCPUs, 256 GB of RAM, and one NVIDIA A100 GPU. AIRA (Toledo et al., 2025) uses OpenAI o3 (OpenAI, 2025) as its backbone LLM and is evaluated with a budget of 24 hours on a machine with 24 vCPUs, 120 GB of RAM, and one NVIDIA H200 GPU. MLEvolve (Du et al., 2025) uses Gemini 3 Pro Preview (Google, 2025) as its backbone LLM and is evaluated with a budget of 12 hours on a machine with 21 vCPUs, 234 GB of RAM, and one NVIDIA H200 GPU.

B. Other experimental results

B.1. Performance on temporal and tabular tasks

Temporal and signal modeling focuses on learning predictive patterns from ordered data such as time series and audio signals (van den Oord et al., 2016; Hershey et al., 2016), commonly addressed using recurrent neural networks such as LSTMs (Hochreiter & Schmidhuber, 1997), temporal convolutional networks (Bai et al., 2018), WaveNet (van den Oord et al., 2016), or spectrogram-based approaches that convert signals to time-frequency representations for processing by image classification models (Hershey et al., 2016). Structured data modeling concerns predictive learning from tabular datasets composed of heterogeneous numerical and categorical variables (Grinsztajn et al., 2022), frequently addressed using gradient-boosted decision trees such as XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018), as well as neural network approaches such as TabNet (Arik & Pfister, 2021). We evaluate AIBuildAI on 3 temporal and signal modeling tasks, including scientific signal prediction (seti-breakthrough-listen, icecube-neutrinos-in-deep-ice) and time-series classification (hms-harmful-brain-activity-classification), as well as 3 structured data modeling tasks, including structured regression (new-york-city-taxi-fare-prediction), materials property prediction (nomad2018-predict-transparent-conductors), and general tabular prediction (tabular-playground-series-dec-2021). As shown in Fig. 6c, AIBuildAI outperforms both baselines (MLEvolve and AIRA-dojo) on all 6 tasks, demonstrating its effectiveness across both temporal and structured data modalities.

Fig. 6a illustrates an example workflow of AIBuildAI on a temporal signal classification task: hms-harmful-brain-activity-classification. It requires classifying EEG (Jasper, 1958) recordings into six categories of harmful brain activity (e.g., seizure, lateralized periodic discharges). To address this task, the AIBuildAI manager agent invokes the designer sub-agent to propose six solution candidates based on diverse approaches including EfficientNet (Tan & Le, 2019) spectrogram classifiers, WaveNet (van den Oord et al., 2016) on raw EEG signals, ResNet (He et al., 2016) and TinyViT (Wu et al., 2022) variants, as well as dual-branch hybrids. The manager reviews the six plans, selects a subset of three candidates to proceed, and invokes the coder and tuner sub-agents to implement and evaluate them. Based on the preliminary results, the manager invokes the designer sub-agent to revise candidates 1 and 2 by adding an EEG branch using double banana montage (Acharya et al., 2016) and a 1D-CNN branch for raw EEG signals, respectively. The coder sub-agent implements the revised designs and the tuner sub-agent retrains the models with adjusted learning rates. The aggregator agent selects the best checkpoint from candidate 1 as the final model.

Fig. 6b depicts the final solution of AIBuildAI on the hms-harmful-brain-activity-classification task in detail. The pipeline takes two inputs, a spectrogram parquet file and a raw EEG parquet file, and processes them through a dual-branch architecture. The spectrogram branch extracts four EEG regions (e.g., left lateral, right lateral), fills NaN values by zero and

clips, applies a log1p transform with Z-score normalization, and resizes the result via bilinear interpolation before feeding it into an ImageNet-pretrained (Deng et al., 2009) EfficientNet-B0 (Tan & Le, 2019) with 4-channel input. The EEG branch applies a double banana montage (Acharya et al., 2016), Butterworth bandpass filtering (Butterworth, 1930) and clipping, then reshapes and duplicates the signal into 3 channels and resizes via bilinear interpolation before passing it through a separate EfficientNet-B0 with 3-channel input. A fusion MLP head combines the feature representations from both branches to produce 6-class probabilities over the six brain activity categories.

C. Detailed discussion

The strong performance of AIBuildAI can be attributed to its sub-agent architecture for iteratively refining multiple solutions in parallel. Refining a candidate solution involves heterogeneous operations, such as designing modeling strategies, writing correct code, and training models to achieve optimal performance, each of which requires multi-step reasoning through iterative LLM calls and tool interactions. For example, producing correct code typically requires repeatedly writing code, executing it, inspecting errors, and refining the implementation. Combining all these operations together leads to extensive context lengths that are difficult for a single agent to handle. In contrast, AIBuildAI uses sub-agents for these operations, leveraging the property that one operation does not need to know the details of other operations. For example, a coder sub-agent maintains a coding-focused working context that tracks prior attempts and avoids repeating mistakes, while the tuner sub-agent only needs the final correct code from the coder and its own context focused on optimizing training performance. By decomposing solution refinement into specialized sub-agents, AIBuildAI keeps each agent focused on a well-defined scope, avoiding the context explosion that would arise from maintaining all intermediate reasoning steps in a single agent.

AIBuildAI also differs from prior LLM-based automated AI model development approaches such as AIRA and MLEvolve, which formulate this task as a search over code solutions. In these systems, each modification to a candidate solution is implemented through a single LLM call that directly transforms one code repository into another. As a result, system performance is critically limited by what can be accomplished in a single LLM call, constraining the complexity of modifications at each step. In contrast, AIBuildAI instantiates each operation as a specialized sub-agent rather than a single LLM call. Each sub-agent (designer, coder, tuner) is itself an LLM-based agent with its own prompt, context, and tool-use capabilities, and may execute multiple rounds of reasoning and tool interaction through iterative LLM calls within a single agent invocation. This substantially increases the expressiveness and effectiveness of each operation: a sub-agent can conduct multi-step reasoning and iteratively use tools, whereas a single LLM call must produce a complete modification in one shot. This shift from single-call transformations to multi-step, agent-driven operations is a key factor in AIBuildAI's ability to produce high-quality solutions across diverse tasks.

Despite these advantages, a limitation of AIBuildAI is that the use of multiple interacting sub-agents increases token consumption costs compared with baseline approaches, as each sub-agent inherently conducts multiple rounds of LLM calls. In addition, repeated communication is required across the manager and sub-agent hierarchy, further increasing the cost. AIBuildAI currently uses a single high-capability language model for all sub-agents, but not all LLM calls require the same level of capability. A promising direction is dynamic LLM routing (Ding et al., 2024; Wang et al., 2025), in which simpler requests (e.g., routine code modifications or configuration updates) are delegated to smaller, cheaper models while only the most challenging operations (e.g., novel architecture design or complex debugging) invoke the most capable model. Such routing mechanisms could potentially reduce costs without sacrificing performance.

Several directions for future work emerge from this study. First, the current evaluation follows the MLE-Bench protocol of a single GPU with a 24-hour time budget, which provides a standardized and reproducible setting. However, real-world AI development tasks, particularly in the era of large language models, often require substantially more computational resources, including multi-GPU training, distributed data parallelism, and longer training horizons spanning days or weeks. As scaling laws have shown that training compute requirements for state-of-the-art models grow rapidly with model and dataset size (Hoffmann et al., 2022), extending AIBuildAI to reason about resource allocation becomes increasingly important. For example, adapting modeling strategies based on available compute, automatically selecting distributed training configurations such as parallelism strategies (Zheng et al., 2022; Zhu et al., 2025), or dynamically adjusting time budgets, would improve its applicability to large-scale, production-grade AI development. Second, AIBuildAI currently relies entirely on the parametric knowledge embedded in the underlying language model and a general-purpose web search tool. Incorporating a structured AI task knowledge base, organized as a retrievable collection of modeling strategies, preprocessing techniques, and architecture patterns extracted from the literature and prior successful runs, could improve

both the quality and consistency of the system's decisions (Wang et al., 2024). Such a knowledge base could be organized hierarchically so that sub-agents can efficiently access domain-relevant knowledge, and could be dynamically updated after each run with newly discovered strategies and their empirical outcomes, enabling the system to accumulate experience and improve over time (Xu et al., 2025).

More broadly, systems like AIBuildAI could reshape how domain scientists interact with AI across the physical and life sciences, where data are increasingly abundant but AI expertise remains scarce. In genomics, single-cell and spatial transcriptomics experiments now profile thousands of genes across millions of cells, yet building the deep learning models needed for cell-type annotation or perturbation-response prediction still requires specialized bioinformatics skills that most experimental biologists lack (Peidli et al., 2024). In materials science and chemistry, self-driving laboratories can generate thousands of experiments autonomously (Abolhasani & Kumari, 2023), but the surrogate models that guide experimental design must be manually constructed and tuned by dedicated ML engineers. In drug discovery, virtual screening of compound libraries against protein targets demands custom scoring functions and molecular representations that require computational chemistry expertise to build and validate (Sadybekov & Katritch, 2023). By automating the full AI model development pipeline, from strategy design through implementation to training and optimization, systems like AIBuildAI could enable scientists in these domains to rapidly build high-performing predictive models directly from their experimental data, accelerating the extraction of scientific insights without requiring deep AI engineering expertise. As experimental throughput continues to outpace analytical capacity across the sciences, the ability to automatically transform raw data into trained, deployable AI models may become as transformative for scientific discovery as the instruments that generate the data themselves.

D. Detailed Benchmark Results and Task Workflows

In this section, we provide extended experimental results and detailed walkthroughs of the **AIBuildAI** system across diverse domains. These figures illustrate the system's ability to handle varying levels of task complexity and its effectiveness in designing, implementing, and refining solutions for vision, language, and temporal data.

- **MLE-Bench Performance:** Figure 2 displays the overall leaderboard standings and performance across different task difficulty levels.
- **Computer Vision:** Figure 3 and Figure 4 detail the sub-agent workflows for image classification, detection, and segmentation, highlighting the ensemble strategies and architectural choices made by the system.
- **Natural Language Processing:** Figure 5 demonstrates the system's capability in language modeling and sequence tasks, showcasing a refined BERT-based approach for word imputation.
- **Temporal and Tabular Data:** Figure 6 explores complex signal processing, such as EEG classification, using dual-branch architectures and specialized preprocessing.

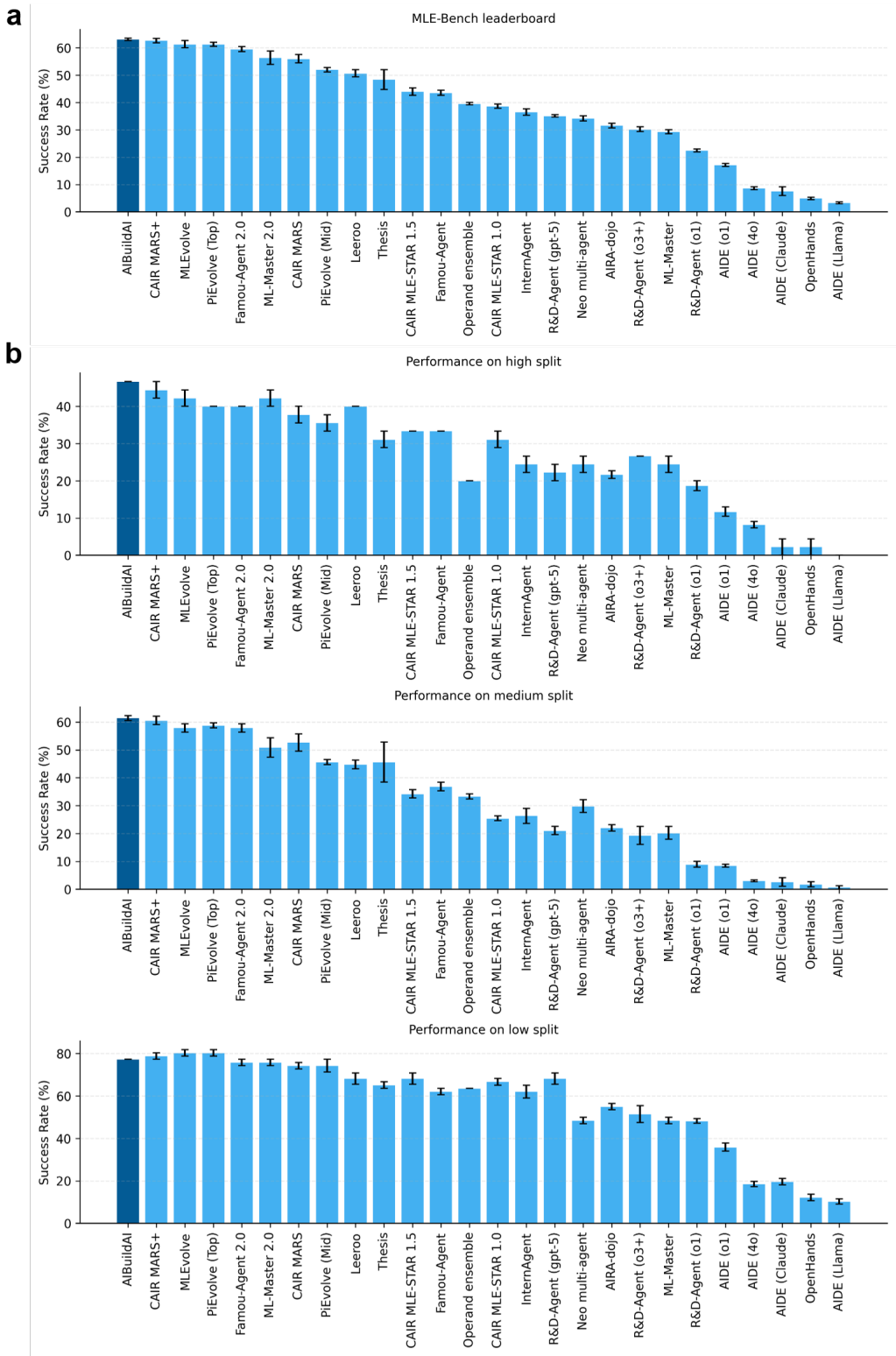


Figure 2. **AIBuildAI ranks first on the MLE-Bench leaderboard. a**, Overall medal rate of AIBuildAI compared with 26 recent baseline methods (or method variants) on MLE-Bench as of the March 18, 2026 leaderboard. AIBuildAI achieves the top overall position with a medal rate of 63.1%, surpassing prior systems including MARS, Famou-Agent, ML-Master, Leeroo, InternAgent, R&D-Agent, AIDE, AIRA-dojo, and MLEvolve. **b**, Medal rates of AIBuildAI and baseline methods on the three complexity splits of MLE-Bench. AIBuildAI ranks first on both the medium- and high-complexity splits, demonstrating that its performance advantage becomes more pronounced on more challenging AI tasks.

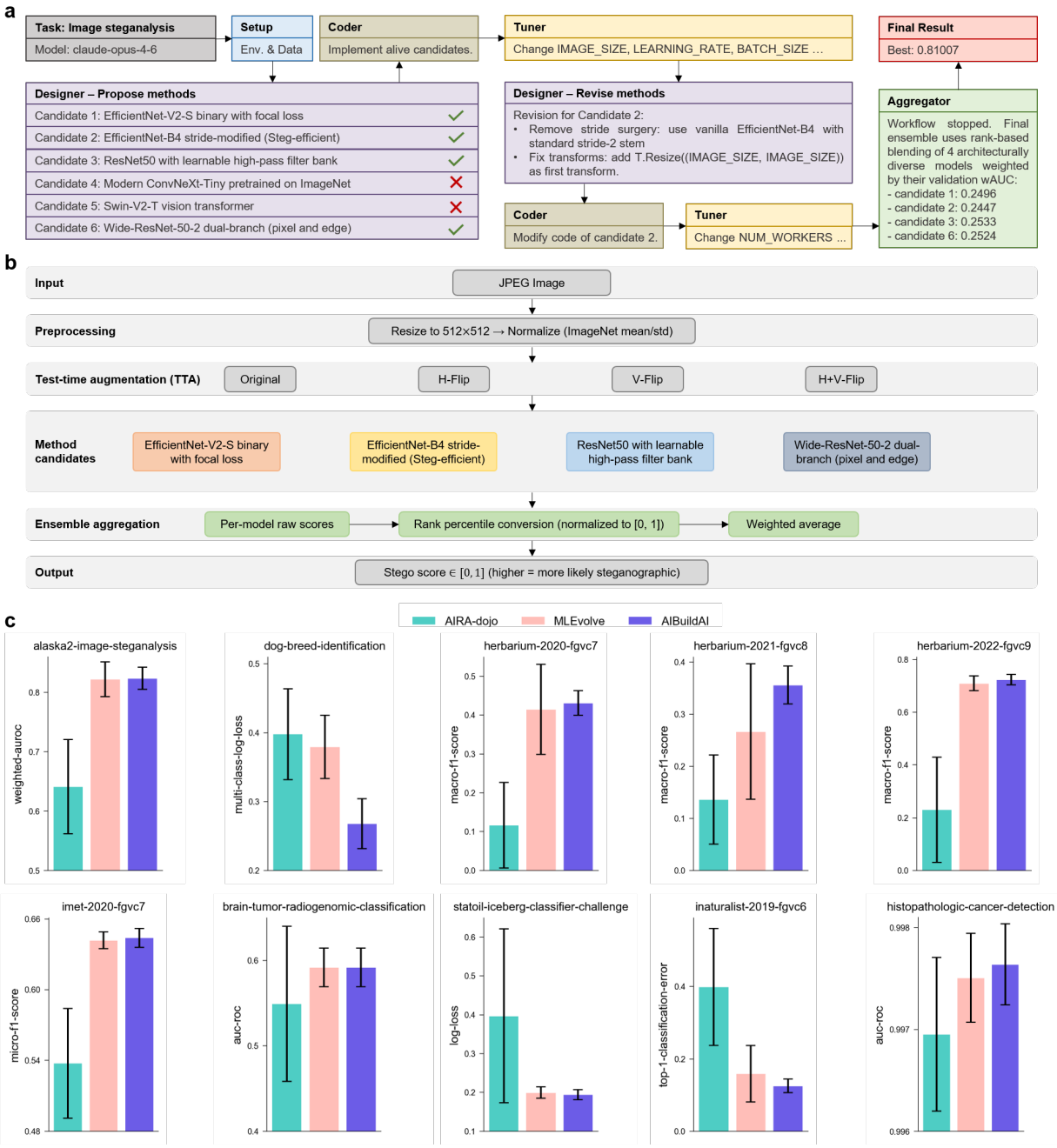


Figure 3. **AIBuildAI outperforms baseline methods on image classification tasks. a,** Example workflow on the alaska2-image-steganalysis task, a steganalysis challenge to predict the likelihood that an image contains a hidden message. The manager agent invokes the designer sub-agent to propose six solution candidates based on diverse architectures including EfficientNet, ResNet, ConvNeXt, Swin Transformer, and Wide-ResNet variants. The manager reviews the six plans, selects four to proceed, and invokes the coder sub-agent to implement each plan and the tuner sub-agent to optimize training parameters and obtain validation results. Based on these results, the manager invokes the designer to revise candidate 2 by removing stride surgery and reverting to vanilla EfficientNet-B4, which then passes through the coder and tuner again. The aggregator constructs a rank-based ensemble of the four models, achieving a score of 0.8101 on the private test set. **b,** Final solution in detail. The pipeline takes an image as input, resizes and normalizes it using ImageNet statistics. Test-time augmentation produces four views (original, H-flip, V-flip, H+V-flip). Each view is passed through four ensemble models, namely EfficientNet-V2-S with focal loss, EfficientNet-B4, ResNet50 with a learnable high-pass filter bank, and Wide-ResNet-50-2 with a dual-branch architecture. Per-model raw scores are converted to rank percentiles and combined via weighted averaging to produce a final stego score. **c,** Performance across 10 image classification tasks spanning natural image recognition, fine-grained species classification, medical imaging, data security, and object recognition. AIBuildAI consistently outperforms baseline systems AIRA-dojo and MLEvolve across various visual domains and evaluation metrics.

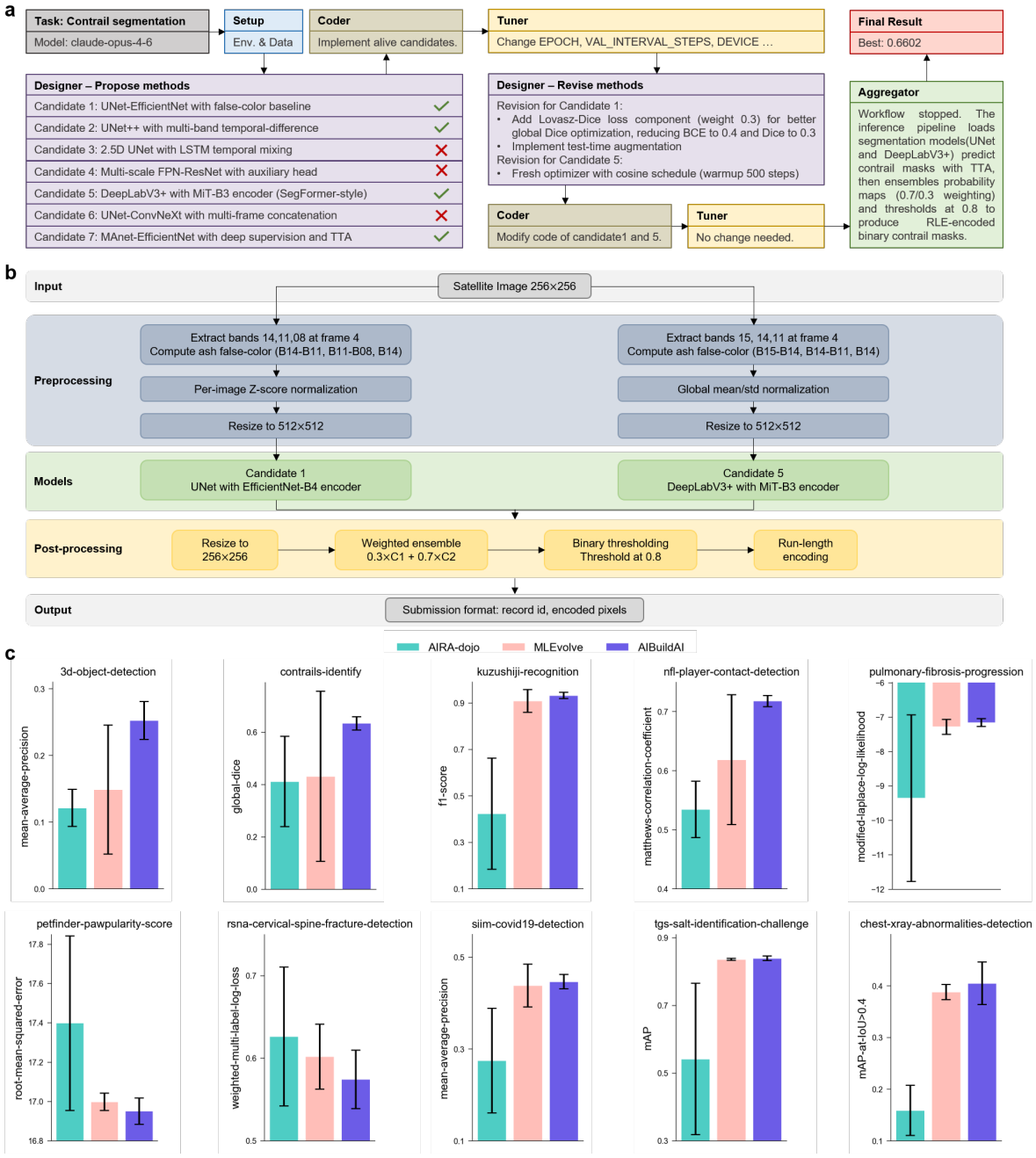


Figure 4. **AIBuildAI surpasses baseline methods on detection, segmentation, and video understanding tasks. a,** Example workflow on the contrails-identify task, which aims to generate masks of aircraft condensation trails from satellite imagery. The manager agent invokes the designer sub-agent to propose seven solution candidates based on diverse segmentation architectures including UNet, FPN-ResNet, DeepLabV3+, and ConvNeXt variants. The manager reviews the plans, selects four candidates to proceed with, and invokes the coder and tuner sub-agents to implement, train and evaluate them. Based on the results, the manager invokes the designer to revise candidates 1 and 5 by adding a Lovász-Dice loss component and a fresh optimizer with cosine schedule, respectively. The coder and tuner implement and retrain the revised models. The aggregator constructs a weighted ensemble of the two strongest candidates, achieving a Dice score of 0.660 on the private test set. **b,** Final solution in detail. The pipeline takes a satellite image and processes it through two parallel branches. The UNet branch (EfficientNet-B4 encoder) computes ash false-color composites from selected satellite bands with per-image Z-score normalization. The DeepLabV3+ branch uses a MiT-B3 encoder, a hierarchical vision transformer, with a different set of bands and global normalization. Each model produces a per-pixel probability map indicating contrail presence. The two maps are combined with weighted averaging, thresholded, and encoded in run-length format. **c,** Performance across 10 visual perception tasks spanning object detection, semantic segmentation, medical imaging, character recognition, regression, and video-based activity recognition. AIBuildAI consistently outperforms baseline systems AIRA-dojo and MLEvolve across diverse spatial and temporal modalities.

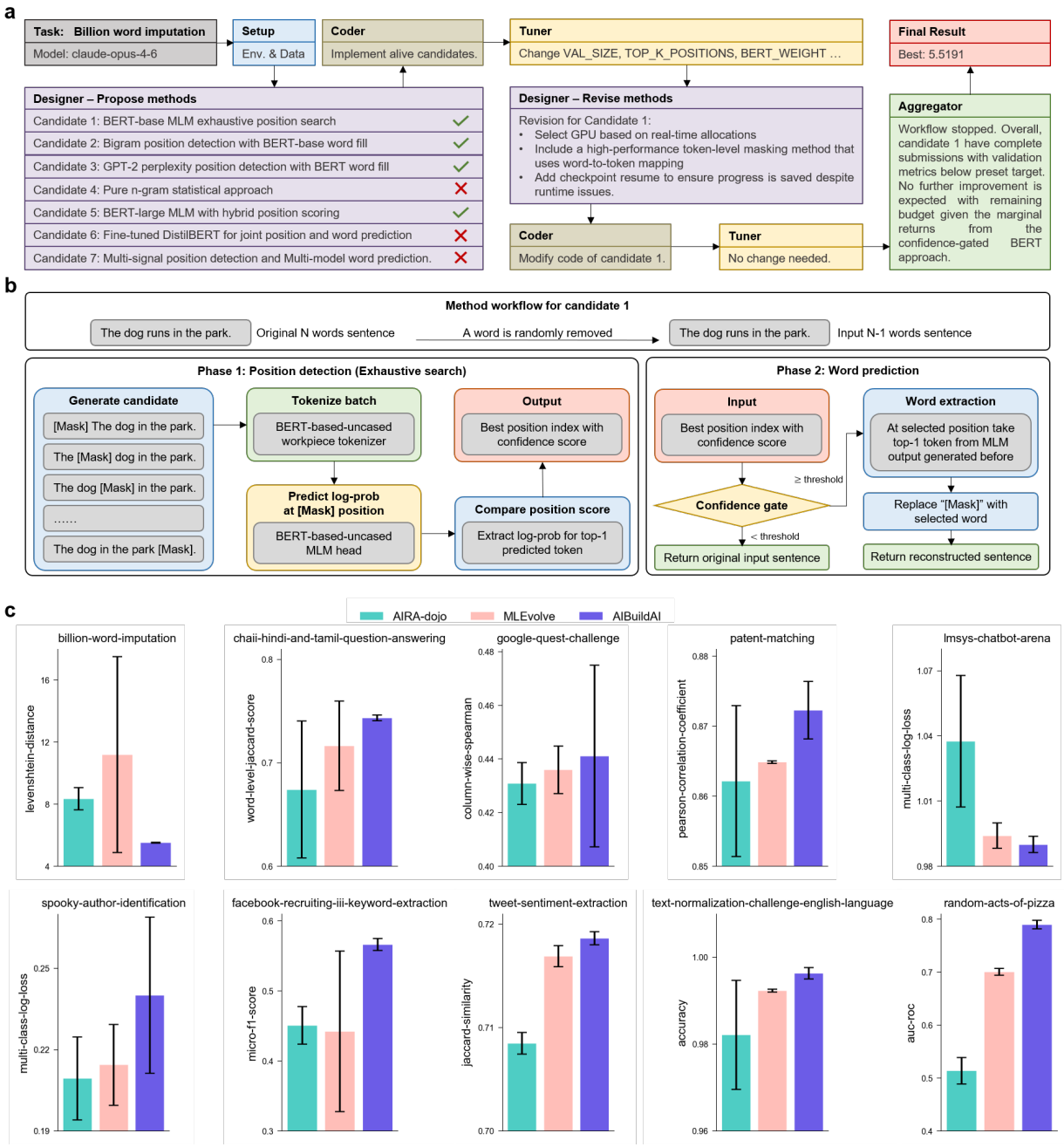


Figure 5. **AIBuildAI outperforms baseline methods on language understanding and generation tasks. a,** Example workflow on the billion-word-imputation task, which aims to predict a missing word at an unknown position in a text paragraph. The manager agent invokes the designer sub-agent to propose seven solution candidates based on diverse approaches including BERT, DistilBERT, n-gram, and multi-model methods. The manager reviews the plans, selects a subset to proceed, and invokes the coder and tuner sub-agents to implement and evaluate them. Based on the results, the manager invokes the designer to revise candidate 1 by incorporating a token-level masking method with word-to-token mapping. The coder and tuner implement and retrain the revised model. The aggregator selects candidate 1 based on validation metrics, achieving a Levenshtein distance of 5.519 on the private test set. **b,** Final solution in detail. Given an input sentence with a missing word, the pipeline generates candidates by inserting a mask token at every possible position. Each candidate is tokenized and passed through a BERT-base-uncased model, which computes log-probabilities over the vocabulary at the masked position. The position with the highest predicted-word confidence is selected. A confidence gate then determines whether to replace the mask with the predicted word and return the reconstructed sentence, or retain the original input unchanged. **c,** Performance across 10 language-related tasks spanning text classification, question answering, information extraction, sequence-to-sequence generation, word imputation, and sentiment analysis. AIBuildAI consistently outperforms baseline systems, demonstrating strong capability across diverse language understanding and generation tasks.

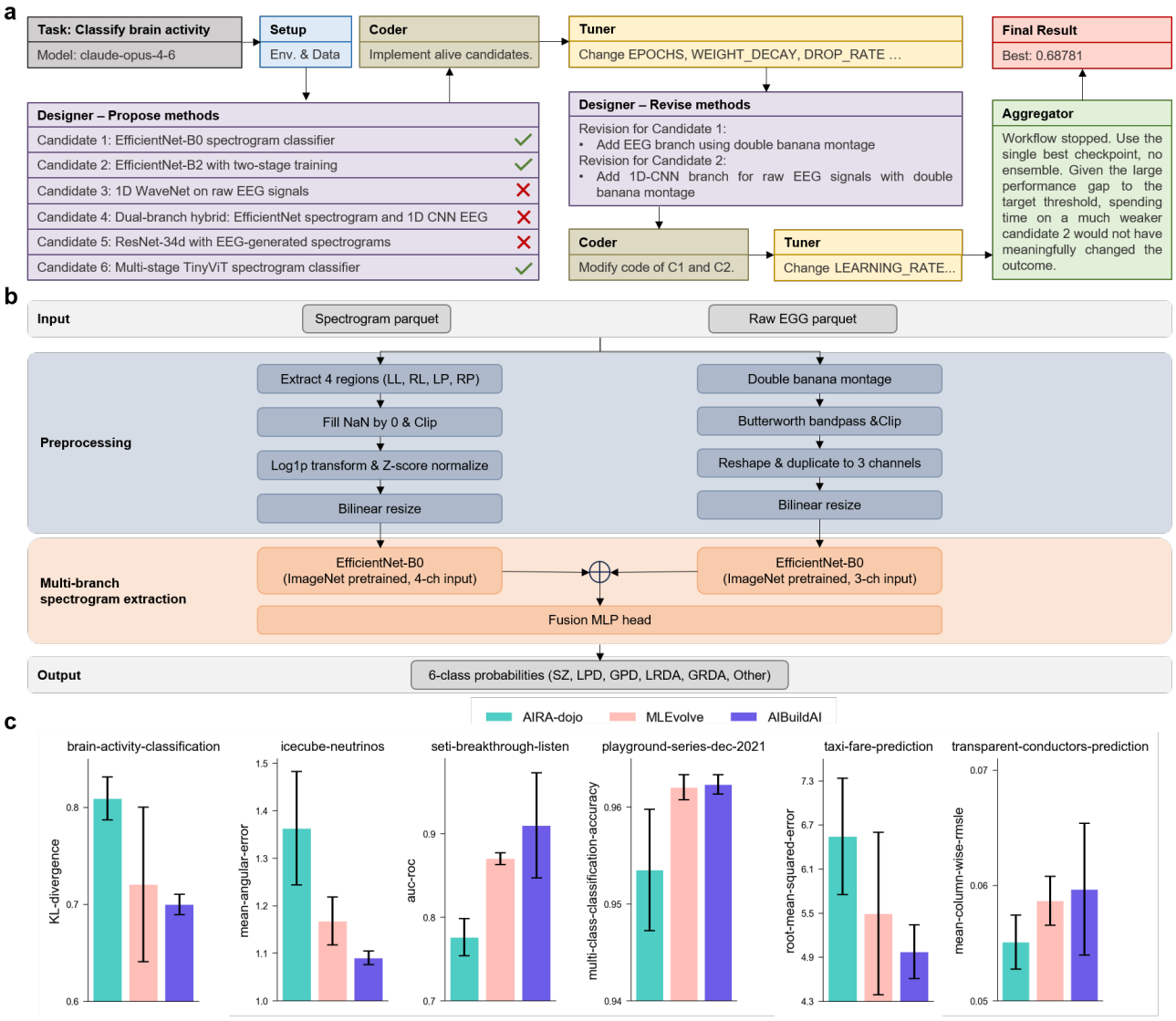


Figure 6. **AIBuildAI surpasses baseline methods on temporal and tabular tasks. a,** Example workflow on the hms-harmful-brain-activity-classification task, which requires classifying EEG recordings into six categories of harmful brain activity (e.g., seizure, lateralized periodic discharges). The manager agent invokes the designer sub-agent to propose six solution candidates based on diverse approaches including EfficientNet spectrogram classifiers, WaveNet on raw EEG, ResNet, TinyViT, and dual-branch hybrids. The manager reviews the plans, selects a subset to proceed, and invokes the coder and tuner sub-agents to implement and evaluate them. Based on the results, the manager invokes the designer to revise candidates 1 and 2 by adding an EEG branch with double banana montage and a 1D-CNN branch for raw signals, respectively. The coder and tuner implement and retrain the revised models. The aggregator selects the best checkpoint from candidate 1 as the final model. **b,** Final solution in detail. The pipeline takes spectrogram and raw EEG inputs and processes them through a dual-branch architecture. The spectrogram branch extracts four EEG regions (e.g., left lateral, right lateral), fills missing values, applies log-transform with Z-score normalization, and feeds the result into an ImageNet-pretrained EfficientNet-B0. The EEG branch applies a double banana montage and Butterworth bandpass filtering, then passes the processed signal through a separate EfficientNet-B0. A fusion MLP head combines both branches to produce 6-class probabilities. **c,** Performance across 3 temporal signal modeling tasks (EEG classification, neutrino detection, radio signal search) and 3 structured tabular prediction tasks (regression, materials property prediction, general tabular prediction). AIBuildAI consistently outperforms baseline systems AIRA-dojo and MLEvolve across both temporal and structured data modalities.