AUTOBRICKS: LLM-DRIVEN AUTOMATED BRICK BUILDING FOR SPATIO-TEMPORAL FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing high-performing spatio-temporal forecasters requires expertise in matching architectures to data topology, while off-the-shelf LLMs can introduce interface errors and tensor mismatches when generating forecasting code. We address this gap through retrieval-augmented modular assembly over a standardized brick registry, with a frozen LLM proposing architectures from task descriptions and retrieved experience. Our framework, **AutoBricks**, provides 38 operators with standardized invocation interfaces. A dual memory mechanism retrieves successful architectural configurations and failure trajectories to guide candidate generation, while execution and validation feedback inform subsequent proposals without updating LLM parameters. Experiments across 20 spatio-temporal datasets demonstrate high code executability and up to 15.1% lower MAE than the strongest baselines. AutoBricks also substantially reduces search costs relative to NAS methods. Operator-pool and random-assembly comparisons further demonstrate the effectiveness of guided assembly. Code and demo are available at `https://anonymous.4open.science/r/AutoBricks/`.

1 INTRODUCTION

Spatio-Temporal (ST) forecasting underpins applications from traffic management to environmental monitoring (Farahmand et al., 2023; Aryandoust et al., 2022; Saad et al., 2024), and a rich spectrum of CNN (LeCun et al., 1998), GNN (Kipf & Welling, 2016; Li et al., 2024), and ST Transformer (Shao et al., 2022b) architectures has been developed to capture diverse spatial topologies and temporal dependencies. Yet deploying these models in practice still requires data scientists to hand-pick architectures for each data structure and hand-tune their hyperparameters, raising the barrier to entry for advanced ST forecasting and limiting its scalable adoption.

Large Language Models (LLMs) offer a promising route to automated model development, but generated code can contain hallucinated references, interface incompatibilities, and tensor mismatches (Agarwal et al., 2024; Wang et al., 2025). Agentic coding tools such as Claude Code and Codex support iterative code modification and testing, while autonomous research frameworks extend automation to experimentation. AI Scientist-v2 spans hypothesis generation, experiments, and manuscript preparation (Yamada et al., 2025), and autoresearch refines language-model training code using validation feedback under a fixed training-time budget [1]. Within this landscape, AutoBricks targets ST architecture design, where diverse temporal patterns and spatial structures require task-dependent model choices (Qiu et al., 2024; Li et al., 2025). As illustrated in **Figure 1**, our Retrieval Augmented Modular Assembly paradigm specializes the search through a standardized ST operator registry and retrieval of successful configurations and failure records, with execution checks and forecasting validation evaluating candidate architectures.

Realizing this paradigm presents three coupled challenges. **First, domain-specific fine-tuning entails data preparation and model adaptation costs.** Despite advances in efficient adaptation (Liu et al., 2024; Zhao et al., 2024) and self-play fine-tuning (Chen et al., 2024), ST architecture generation still requires suitable supervision and model updates. **Second, semantic plausibility does not ensure executable correctness** (Wang et al., 2025), as non-existent modules and tensor mismatches can pass syntax checks yet fail at runtime. **Third, retrieval relevance does not ensure task compatibility.**

[1] https://github.com/karpathy/autoresearch

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Although self-reflection (Asai et al., 2024) and structured knowledge augmentation (Jiang et al., 2025) enrich retrieval-based generation, retrieved architectures still require validation against prediction horizons and hardware budgets.

To address these challenges, AutoBricks performs sequential ST architecture search over a standardized brick registry, with a frozen LLM proposing candidates conditioned on the prompt context. The registry standardizes operator interfaces, while a dual memory mechanism retrieves successful architectural anchors as positive priors and historical failure trajectories to discourage incompatible combinations. An ϵ-controlled mixture of random sampling and LLM-guided proposals balances exploration and exploitation, while dual-granularity feedback from execution checks and forecasting validation is summarized into reflective prompt updates that guide subsequent proposals with the LLM frozen.

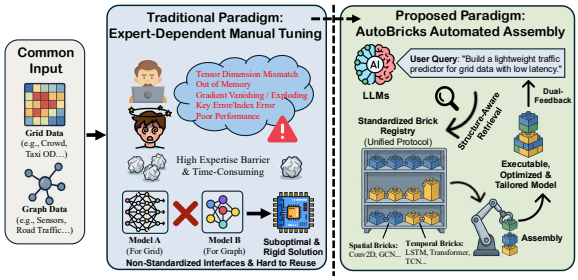


Figure 1: Traditional manual tuning vs. AutoBricks. Left, expert-driven design yields rigid models that are hard to reuse across tasks. Right, AutoBricks lets an LLM assemble executable models from a standardized Brick Registry under retrieval guided control.

Our contributions are as follows. (i) We formulate ST architecture design as retrieval-augmented modular assembly over a registry of 38 standardized operators, enabling a frozen LLM to propose task-dependent architectures. (ii) We establish a dual experience memory that retrieves successful configurations and failure trajectories to guide candidate assembly. (iii) We develop a context refinement mechanism that combines execution diagnostics and forecasting validation with reflective prompt updates to guide subsequent proposals under frozen LLM parameters. (iv) Experiments across 20 ST datasets demonstrate competitive forecasting accuracy and high code executability, with search-cost measurements, operator-pool controls, and random-assembly comparisons examining the efficiency and effectiveness of the search workflow.

2 Preliminaries and Problem Definition

In this section, we provide a unified formulation for spatio-temporal forecasting across heterogeneous data structures and formally define the automated modular assembly problem.

Definition 1 (Spatio-Temporal Forecasting). ST forecasting aims to infer future system states from historical observations. To handle diverse urban sensing data formats (e.g., grid-based traffic speed (Yuan et al., 2024; Jiang et al., 2021) and graph-based sensor readings (Liu et al., 2023b; Li et al., 2023)) within a unified framework, we formalize ST data as a general tensor sequence. Let $\mathcal{X} = \{X_1, X_2, \ldots, X_L\}$ denote a sequence of observations with total length L, where $X_t \in \mathbb{R}^{N \times C}$ represents the system state at time step t. Here, N denotes the number of spatial nodes or grid units, and C represents the feature dimension, like traffic volume or speed. To handle heterogeneous spatial structures under a shared interface, we use $\mathcal{G}$ as a structural argument whose role depends on the data topology:

- **Grid-based Data (Euclidean):** For grid data (e.g., TaxiBJ), spatial relationships are implicit in Euclidean lattice coordinates. We reshape the observation to $X_t \in \mathbb{R}^{H' \times W' \times C}$, where $N = H' \times W'$. Grid-specific operators recover (H', W') from dataset metadata when two-dimensional operations are required and otherwise ignore $\mathcal{G}$.
- **Graph-based Data (Non-Euclidean):** For graph data (e.g., METR-LA), we set $\mathcal{G} = A \in \mathbb{R}^{N \times N}$, where A_{ij} indicates the connection strength or proximity between nodes i and j.

Forecasting Problem Formulation. Given a historical observation sequence $\mathcal{X}_{t-T+1:t}$ of length T, the goal is to learn a mapping function f_θ to predict the future H time steps:

$$\hat{\mathcal{Y}}_{t+1:t+H} = f_\theta(\mathcal{X}_{t-T+1:t}, \mathcal{G}) \tag{1}$$

where θ denotes the trainable parameters of the forecasting model.

To automate the design of f_θ rather than relying on manual craftsmanship, we conceptualize the deep learning model as a Directed Acyclic Graph (DAG) composed of standardized components.

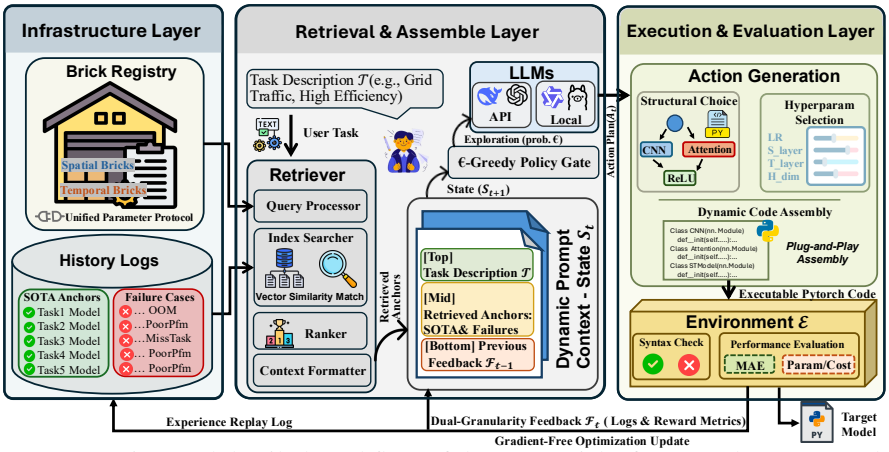


Figure 2: An overview and detailed workflow of the AutoBricks framework. A user task triggers prior retrieval, guiding the LLM to assemble modules from the Brick Registry. Execution feedback then drives iterative optimization.

Definition 2 (Brick Registry). Let $\mathcal{B} = \{b_1, b_2, \ldots, b_M\}$ be the registry of atomic operators, where each brick b_i (e.g., `ConvLSTM`, `GCN`, `TemporalAttention`) encapsulates a standardized input-output interface compatible with the shared structural argument $\mathcal{G}$.

Definition 3 (Assembly Action). An assembly action is defined as a discrete configuration sequence $\mathcal{A}$, which specifies operators from $\mathcal{B}$, their topological connections, fusion strategies at multi-parent nodes, and corresponding hyperparameters such as hidden dimensions.

Automated Assembly Formulation. The Automated Modular Assembly problem seeks an optimal configuration $\mathcal{A}^*$ within a constrained search space Ω. Let $\hat{\theta}_{\mathcal{A}}$ denote the model weights obtained by training architecture $\Phi_{\mathcal{A}}$ on $\mathcal{D}_{train}$ under the prescribed training procedure. The objective is to minimize validation loss subject to resource limits $\mathcal{K}$.

$$\mathcal{A}^* = \arg\min_{\mathcal{A}\in\Omega} \mathbb{E}_{\mathcal{D}_{val}} \left[\mathcal{L}(f_{\Phi_{\mathcal{A}},\hat{\theta}_{\mathcal{A}}}(\mathcal{X},\mathcal{G}),\mathcal{Y}) \right] \quad \text{s.t.} \quad \text{Cost}(f_{\Phi_{\mathcal{A}},\hat{\theta}_{\mathcal{A}}}) \leq \mathcal{K}. \tag{2}$$

Here $\mathcal{L}$ denotes prediction error and $(\mathcal{X},\mathcal{Y}) \sim \mathcal{D}_{val}$ denotes a validation input-target window. The resource limit $\mathcal{K}$ is enforced as a hard feasibility constraint, while Eq. 9 ranks feasible candidates using prediction accuracy and parameter cost. Forecasting weights are optimized by gradient descent, while a frozen LLM guides the outer search through prompt updates.

3 AutoBricks Framework

Figure 2 depicts the AutoBricks pipeline with three layers. The infrastructure layer provides a standardized brick registry and dual experience memory. The retrieval and assembly layer retrieves architectural anchors and failure records, then generates candidates through an ϵ-controlled mixture of random sampling and LLM-guided assembly. The execution and evaluation layer instantiates candidates as PyTorch models, checks execution, and trains valid candidates, with execution diagnostics and forecasting validation informing prompt updates for the next iteration.

3.1 Standardized Bricks and Dual Memory

Standardized Brick Registry. To confine open-ended code generation within a controllable decision process, we abstract every operator b_i as a typed mapping

$$\mathbf{Z} = b_i(X, \mathcal{G}; \Phi_i), \tag{3}$$

over a unified 4D tensor $X \in \mathbb{R}^{B\times T\times N\times C_{in}}$, where $\mathcal{G}$ is the shared structural argument and Φ_i is the operator-specific hyperparameter set. Graph bricks use $\mathcal{G}$ for message passing, whereas grid bricks ignore it and recover (H', W') from dataset metadata when required.

We curate 38 representative ST modules from recent top-tier venues as shown in **Table 1**, covering both established architectures and recent SOTA. For modules whose original code was missing

(ACFM (Liu et al., 2018b), PCRN (Zonoozi et al., 2018)) or written in legacy TensorFlow (ST-ResNet (Zhang et al., 2017), DMVST-Net (Yao et al., 2018)), we performed rigorous PyTorch refactoring, decoupling nested logic from the original papers and encapsulating core computational units as independent `nn.Module` subclasses for fair benchmarking.

Heterogeneous operators such as grid-based ConvLSTM and graph-based GAT expose different interfaces. We standardize their invocation through two interface conventions. All operators accept and return 4D tensors with axis layout (B, T, N, C), using automatic axis-folding and reshaping for modules without an explicit temporal or spatial axis. The structural argument $\mathcal{G}$ is passed through a shared slot, which grid operators ignore and graph operators use for message passing. This protocol provides a common invocation signature across data topologies, while candidate compositions must still satisfy operator-specific dimension and topology requirements.

Table 1: Overview of the AutoBricks standardized brick registry. † denotes re-implemented from scratch, $*$ denotes ported from TensorFlow to PyTorch, and the remainder are decomposed and standardized from original open source codebases.

Topology	Grid-based		Graph-based	
Category	**Bricks**	**Source**	**Bricks**	**Source**
Spatial	CNN†	CNN (LeCun et al., 1998)	GAT†	GAT (Veličković et al., 2017)
	GatedCNN*	STDN (Yao et al., 2019)	GCN†	GCN (Kipf & Welling, 2016)
	Local CNN *	DMVST-Net (Yao et al., 2018)	AdaptiveGCN	GWNet (Wu et al., 2019)
	ResConv*	ST-ResNet (Zhang et al., 2017)	ChebGCN	STGCN (Yu et al., 2018)
	ResPlusConv*	DeepSTN+ (Lin et al., 2019)	DiffusionGCN	D^2STGNN (Shao et al., 2022c)
	SFEncoder	ST-GSP (Zhao et al., 2022)	GumbelGraph	GTS (Shang et al., 2021)
	SpatialResNet†	ACFM (Liu et al., 2018b)	MixHopGCN	MTGNN (Wu et al., 2020)
	SpatialTransformer	PromptST (Zhang et al., 2023b)	SpatialAttention	STTN (Xu et al., 2020)
	TimeDisConv †	PCRN (Zonoozi et al., 2018)		
	ViTGCN	MC-STL (Zhang et al., 2023a)	SpatialEmbedding	STID (Shao et al., 2022a)
Temporal	AttentionConvLSTM†	ACFM (Liu et al., 2018b)	GRU†	GRU (Cho et al., 2014)
	AttentionGRU†	MC-STL (Zhang et al., 2023a)	LSTM†	LSTM (Hochreiter & Schmidhuber, 1997)
	AttentionLSTM*	STDN (Yao et al., 2019)	GLUTCN	STGCN (Yu et al., 2018)
	BiTransformer	ST-GSP (Zhao et al., 2022)	DilatedTCN	TCN (Bai et al., 2018)
	ConvLSTM†	ConvLSTM (Shi et al., 2015)	DilatedInceptionTCN	MTGNN (Wu et al., 2020)
	EarlyFusionCNN*	DeepSTN+ (Lin et al., 2019)	ResidualMLP	STID (Shao et al., 2022a)
	LSTM†	LSTM (Hochreiter & Schmidhuber, 1997)	TemporalAttention	STTN (Xu et al., 2020)
	PyramidalConvGRU†	PCRN (Zonoozi et al., 2018)	Transformer	Transformer (Vaswani et al., 2017)
	TemporalPrompt	PromptST (Zhang et al., 2023b)	WaveNet	GWNet (Wu et al., 2019)
	TemporalResNet*	ST-ResNet (Zhang et al., 2017)		

Dual Experience Memory. The brick registry defines the available operators, while a dual memory bank $\mathcal{M} = \mathcal{M}^+ \cup \mathcal{M}^-$ guides architecture search. Positive memory stores configurations that improve the best reward observed so far, while negative memory records unsuccessful trajectories, including execution failures and non-improving outcomes. Retrieved records provide positive design anchors and failure information for subsequent proposals. For explicit candidate filtering, let $\mathcal{R}^-_{t,\mathrm{str}} \subseteq \mathcal{R}^-_t$ denote retrieved records associated with operator-interface or topology incompatibilities, excluding resource failures such as OOM and predictive underperformance. These records define the exclusion set

$$\mathcal{C}^-_t = \{\mathcal{A} \in \Omega \mid \exists h \in \mathcal{R}^-_{t,\mathrm{str}},\ \mathrm{Sim}(\Phi_{\mathcal{A}}, \Phi_h) > \tau\}. \tag{4}$$

Here Φ_h denotes the architecture recorded in h, and Sim measures Jaccard overlap between brick-name sets. Candidates in $\mathcal{C}^-_t$ are rejected before training, while other negative records remain in the retrieved context.

3.2 Retrieval Augmented Action Generation

Structure Aware Retrieval. At each step t, we construct a context state S_t from the task description and retrieved architectural records. We vectorize the user task $\mathcal{T}$ and historical trajectories using TF-IDF (Sparck Jones, 1972; Salton & Buckley, 1988), where each historical record is serialized from its architecture description and evaluation feedback.

$$\mathcal{R}^+ = \underset{h \in \mathcal{M}^+}{\text{Top-}K} \frac{v(\mathcal{T})^\top v(h)}{\|v(\mathcal{T})\|\|v(h)\|}, \qquad \mathcal{R}^- = \underset{h \in \mathcal{M}^-}{\text{Top-}K} \frac{v(\mathcal{T})^\top v(h)}{\|v(\mathcal{T})\|\|v(h)\|}. \tag{5}$$

The prompt S_t combines $\mathcal{T}$, retrieved records, and retained feedback to condition the LLM proposal. Empty memories yield no retrieved records. Candidate filtering excludes architectures in $\mathcal{C}^-_t$. For the LLM branch, the distribution conditional on passing this filter is

$$q_t(\mathcal{A} \mid S_t) \propto P_{\theta_{\mathrm{LLM}}}(\mathcal{A} \mid S_t) \cdot \mathbb{I}\left(\mathcal{A} \notin \mathcal{C}^-_t\right), \tag{6}$$

where $\mathbb{I}(\cdot)$ is the indicator function; normalization requires positive retained probability mass. Retrieved records guide generation, similarity filtering excludes candidates, and execution checks assess tensor compatibility and runtime feasibility.

Mixture Policy for Exploration and Exploitation. To explore beyond retrieved anchors, we mix random sampling with LLM proposals conditioned on the task, retrieved records, and recent feedback. Candidate generation follows a time-decaying mixture policy

$$\pi_{mix}(\mathcal{A}_t \mid S_t) = \epsilon_t \cdot \pi_{rand}(\mathcal{A}_t) + (1 - \epsilon_t) \cdot \pi_{LLM}(\mathcal{A}_t \mid S_t), \tag{7}$$

where π_{rand} samples from Ω and π_{LLM} is the frozen LLM conditioned on S_t. The exploration rate follows an exponential schedule $\epsilon_t = \epsilon_{start} \cdot \gamma^t$, gradually shifting from random sampling toward

context-guided proposals. A candidate sampled from either branch is subsequently subjected to the same exclusion filter and execution checks.

Chain of Thought Reasoning. Directly mapping S_t to a full assembly $\mathcal{A}_t$ is difficult because the search is combinatorial in both operator choice and hyperparameters. We therefore prompt the LLM to first emit a chain of thought Z that plans the high-level topology (for example, detect periodicity, then pick a periodicity brick, then choose an embedding layer) before instantiating the concrete configuration. This factorizes generation as

$$P(\mathcal{A}_t \mid S_t) = \sum_{Z} P(\mathcal{A}_t \mid Z, S_t)\, P(Z \mid S_t), \tag{8}$$

which exposes the intermediate design for inspection alongside the selected operators, their connections, and hyperparameters.

3.3 Execution Environment and Textual Gradients

Dynamic Instantiation. The action plan $\mathcal{A}_t$ is a discrete symbol sequence. We define an instantiation map $\Psi : \Omega \to \mathcal{F}_{\text{torch}}$ that projects it into a differentiable computation graph in two steps. Topology parsing traverses the DAG in $\mathcal{A}_t$, builds an `nn.ModuleList` from its connections, applies the specified fusion strategy at multi-parent nodes, and injects $\mathcal{G}$ in the form appropriate to the data type. Then a sandbox forward pass is performed on the input tensor X, providing a runtime check that detects tensor mismatches and execution failures before the candidate proceeds to full model training.

Dual Granularity Feedback. Execution errors and validation scores provide distinct feedback, so we separate the feedback signal into engineering feasibility and predictive quality. In the inner engineering loop, each instantiated model runs a trial inference in the sandbox. If it triggers a `RuntimeError` such as OOM or a shape discrepancy, a log parser translates the Python traceback into a refined natural language error description $\mathcal{L}_{err}$, evaluation terminates immediately, and the candidate skips full training. In the outer loop, surviving models undergo training and validation and receive a composite reward that balances accuracy against resource cost,

$$R(\mathcal{A}_t) = \frac{\alpha}{\text{MAE}(f_{\mathcal{A}_t}, \mathcal{D}_{val}) + \eta} - \beta \cdot \text{Cost}(f_{\mathcal{A}_t}), \tag{9}$$

where $\eta > 0$ is a numerical-stability constant, $\text{Cost}(\cdot)$ is normalized parameter size, and α, β are balancing coefficients. Failed sandbox evaluations receive $-\rho$ as defined in Appendix E.2, while Eq. 9 ranks surviving candidates within each task.

Policy Update via Textual Gradients. The LLM parameters remain frozen throughout search, while execution and validation outcomes update its prompt context. We use *textual gradients* to denote natural-language guidance derived from these observations, rather than derivatives with respect to model parameters. Letting $\mathcal{F}_t = \langle R(\mathcal{A}_t), \mathcal{L}_{err}, \text{Reasoning}_t \rangle$ denote the feedback tuple at step t, the state update is a reflective experience replay

$$S_{t+1} \leftarrow \text{Update}\big(S_t, \text{Reflect}(\mathcal{F}_t), \mathcal{R}^+_{t+1}, \mathcal{R}^-_{t+1}\big). \tag{10}$$

The $\text{Reflect}(\cdot)$ operator rewrites $\mathcal{F}_t$ into a short natural-language diagnosis. For example, an OOM failure guides subsequent proposals toward lighter operator combinations that pass the similarity filter. After memory updates and retrieval, Update incorporates the reflection and refreshed anchors within the context window. Subsequent proposals are thus conditioned on observed outcomes and retrieved experience, while the underlying LLM parameters remain fixed throughout the search.

4 Experiments

We evaluate AutoBricks along four axes, including prediction accuracy, code generation quality, search efficiency, and economic cost, and benchmark it against three paradigms, namely human-designed models, LLM-based code generation, and automated search frameworks covering both general NAS and ST-specific AutoML. Further analyses of statistical significance, hyperparameter sensitivity, search convergence, and cross-task memory transfer are deferred to **Appendix A–D**.

Table 2: Performance on grid-based topologies. The symbols ⋆ and ‡ denote pre-trained and automated models, respectively. **Dark green** and **gray** shadings highlight the best and second-best results, reported to three decimal places.

Models	Metrics	TaxiBJ	Crowd	Cellular	BikeNYC-I	BikeNYC-II	TaxiNYC-I	TaxiNYC-II	TrafficNJ	TrafficCD	TrafficSH	TrafficHZ	TrafficJN	TrafficTJ	TrafficZZ
HA	**MAE**	29.820	6.790	27.570	3.430	7.170	18.570	15.740	0.774	0.668	0.771	0.342	0.690	0.824	0.857
	RMSE	53.770	17.800	72.940	11.410	15.680	57.070	52.840	1.593	1.120	1.240	0.765	1.380	1.610	1.470
ARIMA	**MAE**	39.530	10.230	40.220	3.860	13.630	20.940	29.560	0.709	0.907	0.769	0.364	0.651	1.590	0.998
	RMSE	56.700	21.870	81.310	12.370	25.010	55.390	62.900	1.300	1.540	1.170	0.803	1.200	2.020	1.780
ConvLSTM	**MAE**	31.277	3.705	14.904	5.525	3.924	19.243	13.221	0.711	0.554	0.748	0.430	0.574	0.773	0.785
	RMSE	45.693	6.219	25.955	9.644	7.338	31.262	24.070	1.223	0.856	1.025	0.752	1.021	1.201	1.237
ST-ResNet	**MAE**	30.870	3.382	14.320	4.980	3.940	17.960	12.060	0.635	0.576	0.723	0.406	0.556	0.714	0.693
	RMSE	45.170	5.355	24.300	8.200	7.180	29.450	22.160	1.030	0.867	1.000	0.669	0.964	1.120	1.030
ACFM	**MAE**	21.590	2.340	12.000	1.670	3.094	11.540	6.390	0.515	0.493	0.566	0.316	0.559	0.574	0.526
	RMSE	37.770	4.170	22.790	3.930	5.990	23.350	14.480	0.888	0.757	0.833	0.575	0.920	0.959	0.839
DMVST-Net	**MAE**	33.181	3.613	15.641	5.588	4.110	20.238	14.042	0.692	0.582	0.789	0.447	0.581	0.799	0.761
	RMSE	46.941	6.236	26.601	10.030	7.507	35.963	27.961	1.224	0.878	1.183	0.802	1.010	1.358	1.234
PCRN	**MAE**	31.112	3.541	12.328	5.157	3.519	19.107	12.563	0.649	0.512	0.791	0.408	0.497	0.767	0.685
	RMSE	45.612	6.158	21.800	9.325	6.682	32.113	23.662	1.156	0.803	1.122	0.738	0.898	1.233	1.118
DeepSTN+	**MAE**	31.137	3.409	14.192	4.966	4.038	17.025	10.935	0.662	0.618	0.707	0.397	0.547	0.689	0.687
	RMSE	45.795	5.200	22.711	7.879	6.939	28.180	20.282	1.034	0.877	0.986	0.630	0.894	1.091	0.984
STDN	**MAE**	28.353	3.339	14.173	5.109	3.901	17.345	11.105	0.678	0.557	0.710	0.409	0.552	0.693	0.689
	RMSE	41.908	4.903	22.680	7.803	6.705	28.078	20.144	1.020	0.790	0.969	0.625	0.902	1.073	0.949
STGSP	**MAE**	28.280	4.560	21.400	1.690	7.380	10.290	10.140	0.491	0.497	0.749	0.331	0.490	0.572	0.505
	RMSE	45.040	7.930	39.990	5.000	14.200	28.130	29.100	0.883	0.776	1.020	0.616	0.882	0.989	0.831
MC-STL	**MAE**	15.830	2.390	10.260	2.050	3.400	9.510	8.500	0.780	0.733	0.720	0.606	0.833	0.856	0.810
	RMSE	29.140	4.750	21.220	4.080	6.260	18.440	16.780	1.130	1.000	1.000	0.842	1.190	1.220	1.140
PromptST	**MAE**	14.540	1.540	7.200	1.570	3.229	6.478	5.526	0.531	0.523	0.523	0.342	0.490	0.532	0.502
	RMSE	27.440	3.520	15.740	4.360	6.804	17.229	14.294	1.045	0.880	0.811	0.754	0.953	0.979	0.851
PatchTST⋆	**MAE**	23.160	5.000	20.560	3.050	6.030	14.560	12.610	0.675	0.602	0.679	0.325	0.645	0.740	0.752
	RMSE	43.660	13.510	56.800	9.970	13.620	44.430	41.040	1.420	1.040	1.130	0.726	1.300	1.490	1.350
UniST⋆	**MAE**	13.950	1.380	*6.500*	*1.270*	2.560	*5.650*	*4.820*	0.421	0.415	0.405	0.236	0.430	0.510	0.482
	RMSE	26.840	3.000	14.290	3.500	5.500	*15.320*	*12.710*	0.845	0.711	0.665	0.530	0.843	0.958	0.832
AutoST‡	**MAE**	16.780	3.623	14.896	2.210	4.369	10.549	9.136	0.489	0.436	0.492	0.255	0.467	0.536	0.545
	RMSE	31.633	9.788	41.153	7.224	9.868	32.191	29.735	1.029	0.754	0.819	0.611	0.942	1.080	0.978
Codex	**MAE**	30.130	5.084	23.334	1.880	8.112	11.421	11.121	0.535	0.546	0.822	0.372	0.557	0.628	0.560
	RMSE	42.423	8.093	39.917	5.092	14.289	28.580	29.218	0.882	0.780	1.025	0.634	0.917	0.993	0.844
Claude Code	**MAE**	27.720	4.174	14.526	1.417	8.492	9.599	8.174	0.508	0.471	0.774	0.306	0.499	0.619	0.577
	RMSE	39.936	6.953	36.530	4.203	15.053	23.186	21.882	0.890	*0.644*	0.999	0.547	0.887	0.953	0.858
AutoBricks (LLaMA3.1)	**MAE**	15.445	1.579	**6.452**	1.302	*2.491*	**5.571**	**4.757**	0.508	**0.387**	*0.405*	**0.234**	*0.426*	0.518	*0.476*
	RMSE	29.381	3.340	**13.556**	3.765	5.445	**15.076**	**12.371**	0.863	**0.626**	*0.641*	**0.526**	*0.806*	0.976	*0.813*
AutoBricks (Qwen2.5)	**MAE**	*13.774*	**1.355**	6.508	**1.229**	2.546	6.041	5.211	*0.410*	0.433	**0.390**	0.244	0.430	0.581	0.534
	RMSE	*26.241*	**2.910**	*14.079*	*3.491*	*5.342*	16.345	13.950	*0.791*	0.740	**0.620**	0.531	0.811	1.053	0.860
AutoBricks (GPT5-mini)	**MAE**	**13.703**	1.525	6.854	1.595	2.561	6.759	5.070	0.448	*0.411*	0.429	0.242	0.428	*0.507*	0.503
	RMSE	**26.203**	3.292	14.391	4.211	5.508	17.698	13.499	0.893	0.681	0.666	0.528	0.808	*0.947*	0.838
AutoBricks (DeepSeek)	**MAE**	15.419	*1.363*	6.608	1.348	**2.353**	5.865	5.370	**0.380**	0.421	0.408	*0.235*	**0.422**	**0.487**	**0.467**
	RMSE	29.453	2.933	14.802	**3.458**	**5.254**	15.916	14.209	**0.715**	0.680	0.647	*0.528*	**0.802**	**0.936**	**0.805**

4.1 Experimental Setup

Datasets. We conduct experiments on **20** ST datasets covering both spatial topologies. The **6 graph-based** benchmarks include the traffic speed datasets METR-LA and PEMS-BAY and the traffic flow datasets PEMS03, PEMS04, PEMS07, and PEMS08 (Yu et al., 2018; Shao et al., 2024). The **14 grid-based** datasets (Yuan et al., 2024) span human mobility and cellular activity (Crowd, Cellular, TaxiBJ, TaxiNYC-I, TaxiNYC-II, BikeNYC-I, BikeNYC-II) and citywide traffic speed across seven cities (TrafficNJ, TrafficCD, TrafficSH, TrafficHZ, TrafficJN, TrafficTJ, TrafficZZ). Detailed dataset statistics are summarized in **Appendix G**.

Baselines and Adaptation. We compare AutoBricks against **40** baselines spanning three categories. **Human designed models** cover statistical baselines HA (Smith & Demetsky, 1997) and ARIMA (Box et al., 2015), grid based deep predictors ConvLSTM (Shi et al., 2015), ST-ResNet (Zhang et al., 2017), ACFM (Liu et al., 2018b), DMVST-Net (Yao et al., 2018), PCRN (Zonoozi et al., 2018), DeepSTN+ (Lin et al., 2019), STDN (Yao et al., 2019), STGSP (Zhao et al., 2022), MC-STL (Zhang et al., 2023a), PromptST (Zhang et al., 2023b), PatchTST (Nie et al., 2022), UniST (Yuan et al., 2024), and graph based predictors STGCN (Yu et al., 2018), DCRNN (Li et al., 2018), ASTGCN (Guo et al., 2019), GWNet (Wu et al., 2019), STSGCN (Song et al., 2020), DGCRN (Li et al., 2023), D^{2}STGNN (Shao et al., 2022c), AGCRN (Bai et al., 2020), MTGNN (Wu et al., 2020), STGODE (Fang et al., 2021), StemGNN (Cao et al., 2020), GTS (Shang et al., 2021), STEP (Shao et al., 2022b), STNorm (Deng et al., 2021), STID (Shao et al., 2022a), STAEformer (Liu et al., 2023a), MegaCRN (Jiang et al., 2023), and TimeMixer (Wang et al., 2024). **LLM-based code generation** is represented by Codex with GPT-Astra-6-pro and Claude Code with Claude Fable 5.1, both prompted to construct forecasting models without the AutoBricks brick registry or retrieved memories. **Automated search frameworks** include the general NAS methods ENAS (Pham et al., 2018) and DARTS (Liu et al., 2018a), the grid oriented AutoST (Li et al., 2020), and the graph oriented AutoSTG (Pan et al., 2021), AutoCTS (Wu et al., 2021), and AutoSTF (Lyu et al., 2025). Baseline adaptations, operator-pool controls, and comparison limitations are detailed in **Appendix H**.

Evaluation Metrics. We evaluate performance across three dimensions. **Prediction Accuracy** utilizes standard metrics including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE), as defined in Shao et al. (2024), while **Model Efficiency** is measured by the number of trainable parameters and average training time per iteration.

Table 3: Performance comparison on graph-based topologies. The symbols ⋆ and ‡ denote pre-trained foundation models and automated search frameworks, respectively. Dark green and gray shadings highlight the best and second-best results.

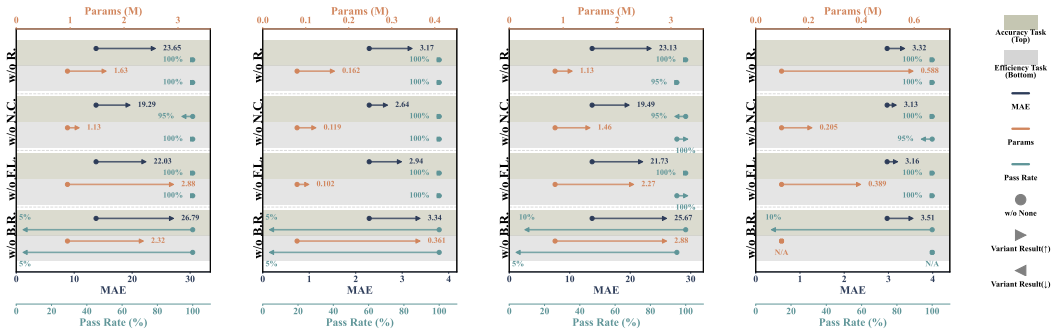
Datasets	METR-LA			PEMS-BAY			PEMS03			PEMS04			PEMS07			PEMS08		
Metrics	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
STGCN	3.21	6.37	8.56%	1.67	3.76	3.78%	15.81	27.55	16.20%	19.93	31.68	13.49%	22.29	35.92	9.65%	16.30	25.50	10.80%
DCRNN	3.02	6.25	8.50%	1.61	3.67	3.72%	15.55	27.37	15.82%	19.67	31.32	13.58%	21.11	34.12	9.10%	15.36	24.18	10.17%
ASTGCN	3.82	7.53	10.42%	1.96	4.37	4.33%	16.16	28.12	17.85%	20.40	32.47	15.11%	23.75	36.58	11.84%	15.90	23.58	11.48%
GWNet	3.02	6.22	8.32%	1.68	3.85	3.67%	14.65	25.43	15.25%	18.71	30.25	13.90%	20.62	33.34	9.35%	14.86	23.64	10.12%
STSGCN	3.61	7.20	9.87%	1.69	3.94	3.87%	15.39	26.95	15.84%	19.23	31.67	13.79%	21.57	35.21	9.12%	14.64	22.44	9.62%
DGCRN	2.90	6.02	8.12%	1.57	3.78	3.60%	14.75	26.21	14.72%	18.99	30.45	12.95%	20.10	33.30	9.31%	14.79	23.94	10.38%
D^{2}STGNN	3.05	5.93	8.10%	1.67	3.56	3.50%	15.02	26.38	15.36%	18.41	29.88	12.65%	19.68	32.49	9.01%	14.60	23.48	10.30%
AGCRN	3.26	6.48	9.00%	1.57	3.64	3.61%	15.19	26.68	14.56%	19.36	30.29	13.31%	20.87	34.53	9.68%	15.25	24.78	10.49%
MTGNN	3.18	6.22	8.22%	1.80	3.74	3.57%	14.93	25.25	15.42%	19.14	29.93	13.38%	21.17	34.31	9.39%	15.41	24.41	10.23%
STGODE	3.05	6.41	8.94%	1.61	3.84	3.68%	15.46	26.19	16.58%	18.91	30.86	13.66%	20.44	33.87	9.09%	14.36	21.75	9.32%
StemGNN	3.68	7.48	10.41%	2.13	4.52	4.57%	15.46	25.10	15.36%	22.95	32.36	16.67%	22.52	36.25	10.43%	14.88	26.43	12.97%
GTS	3.32	6.32	8.69%	1.68	3.92	3.89%	14.88	25.42	15.01%	21.49	31.33	14.94%	22.52	35.58	9.70%	15.07	26.77	10.95%
STEP⋆	3.09	6.04	8.26%	1.43	3.53	3.34%	15.30	26.01	14.48%	18.50	30.03	12.58%	20.55	32.54	8.75%	15.35	23.51	9.58%
STNorm	3.14	6.59	9.02%	1.73	3.82	3.60%	15.35	27.36	16.48%	19.32	32.25	13.09%	20.62	34.90	9.28%	14.40	24.85	9.38%
STID	3.12	6.45	9.14%	1.69	3.66	3.71%	15.52	27.42	16.37%	18.38	29.85	12.52%	19.60	33.76	9.25%	14.37	23.36	9.37%
STAEformer	2.98	6.20	8.22%	1.68	3.89	3.66%	15.84	27.72	16.17%	19.30	30.67	13.32%	20.76	31.27	8.86%	14.72	21.81	9.39%
MegaCRN	2.97	6.26	8.15%	1.65	3.82	3.59%	15.07	26.70	15.27%	19.01	30.12	13.07%	19.74	30.51	8.36%	14.49	21.42	9.21%
TimeMixer	*2.84*	*5.93*	*7.98%*	1.62	3.80	3.57%	14.40	24.28	15.32%	18.47	30.81	13.17%	19.03	31.53	8.01%	14.08	21.99	9.19%
AutoSTG‡	2.98	6.23	8.46%	1.65	3.76	3.80%	15.41	24.41	15.50%	17.54	27.59	11.81%	22.95	35.69	9.97%	13.92	22.05	8.79%
AutoCTS‡	2.98	6.12	8.41%	1.62	3.70	3.74%	12.61	21.04	12.34%	16.40	26.10	11.05%	17.94	28.88	7.63%	12.71	20.27	8.15%
AutoSTF‡	2.87	6.08	8.21%	1.55	3.64	3.63%	*12.38*	20.53	11.82%	15.76	25.60	10.79%	*16.72*	*28.00*	*6.98%*	12.06	19.87	7.84%
Codex	3.83	7.80	11.10%	2.06	4.66	4.88%	15.71	25.97	15.78%	20.58	32.47	14.24%	22.27	35.54	9.72%	15.62	24.70	10.30%
Claude Code	2.90	6.66	8.72%	1.63	3.63	4.48%	15.84	25.42	14.50%	18.56	29.94	14.00%	19.97	31.14	8.95%	13.44	20.73	9.17%
AutoBricks-L	3.29	6.96	9.41%	**1.37**	**3.30**	**3.21%**	14.14	23.05	11.96%	18.75	27.70	12.59%	19.96	30.57	8.92%	13.97	22.11	8.92%
AutoBricks-Q	**2.76**	**5.83**	**7.88%**	1.49	3.42	3.48%	12.55	*20.16*	*10.62%*	*15.38*	*24.07*	*10.33%*	19.89	30.54	8.89%	**11.33**	**18.40**	**7.24%**
AutoBricks-G	3.06	6.47	8.75%	*1.38*	*3.30*	*3.22%*	15.97	24.68	13.51%	15.79	24.52	10.60%	20.58	30.72	9.06%	12.22	19.53	7.80%
AutoBricks-D	2.97	6.29	8.50%	1.43	3.36	3.34%	**10.51**	**17.51**	**8.89%**	**15.32**	**24.04**	**10.28%**	**14.88**	**23.20**	**6.64%**	*11.55*	*18.64*	*7.37%*

Code Generation Quality. We assess syntactic validity using ValidR $= (N_{valid}/N_{total}) \times 100\%$ and single-shot executability using Pass@1 $= (N_{exec}/N_{total}) \times 100\%$. We further use Constraint Satisfaction Rate (CSR) to evaluate whether paired generations follow accuracy-first and efficiency-first instructions. For the N pairs in which both outputs execute, we compute $CSR_{acc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[\mathcal{E}(M_{acc}^{(i)}) < \mathcal{E}(M_{eff}^{(i)})]$ and $CSR_{eff} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[\mathcal{P}(M_{eff}^{(i)}) < \mathcal{P}(M_{acc}^{(i)})]$, where $\mathcal{E}$ and $\mathcal{P}$ denote prediction error and parameter size. ValidR and Pass@1 use all generations, whereas CSR is conditioned on paired executability.

Implementation Details. All experiments run on a server with an Intel Xeon Max 9462 Processor at 2.70GHz and six NVIDIA A100 80GB SXM GPUs. The four LLM backends combine open and closed source options. LLaMA3.1-8B and Qwen2.5-7B are served locally through the vLLM inference framework, while GPT-5-mini and DeepSeek-chat are accessed through their official APIs. System prompt templates are available in the project repository. Generated models are trained with Adam under adaptive learning rate scheduling for up to 100 epochs, with early stopping (patience $= 5$). Following prior practice (Yuan et al., 2024; Zhang et al., 2023b), Crowd and Cellular values are scaled by 1,000 for visualization. The reward uses $\alpha = 1.0$ and $\beta = 0.5$ by default (see **Appendix B**), while final reported architectures are the lowest-validation-MAE candidates per run. Network depth, hidden dimensions, fusion strategies, and other hyperparameters are recommended automatically by AutoBricks for each task.

4.2 Main Results on Forecasting Accuracy

We evaluate forecasting accuracy across 20 datasets, with AutoBricks results averaged over three seeds. **Table 2** reports grid forecasting with input and prediction windows of 6, while **Table 3** reports graph forecasting with windows of 12. Across both tables, at least one AutoBricks backend attains the lowest reported error for each dataset and metric, while no single backend dominates across tasks. On TrafficNJ, DeepSeek achieves an MAE of 0.380, improving over UniST and AutoST by approximately 9.7% and 22.3%, respectively. On TaxiNYC-I, LLaMA3.1 achieves the lowest reported MAE of 5.571. The best AutoBricks variants also outperform Codex and Claude Code across all 14 grid datasets, demonstrating competitive forecasting performance under the evaluated generation and search protocols. On graph datasets, Qwen achieves the lowest MAE on METR-LA and PEMS08, LLaMA3.1 on PEMS-BAY, and DeepSeek on PEMS03, PEMS04, and PEMS07. DeepSeek reaches 10.51 MAE on PEMS03, a 15.1% improvement over AutoSTF (12.38). These results support modular assembly across heterogeneous ST tasks while showing that backend selection remains task dependent. The operator-pool and random-assembly controls in **Table 8** further examine the contributions of operator availability and guided search, while differences in backends and search protocols limit attribution of the main-table gains to individual components.

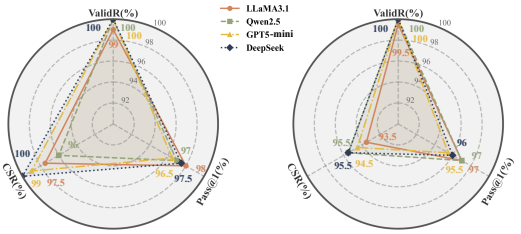


(a) Qwen2.5 on Taxi-iBJ. (b) LLaMA3.1 on PEMS-BAY. (c) GPT5-mini on Taxi-iBJ. (d) DeepSeek on PEMS-BAY.

Figure 5: Ablation analysis across different backend LLMs and spatial structures. The length of the dumbbell line quantifies the performance gap relative to the full AutoBricks framework.

4.3 Study on Generation Quality

The core objective of AutoBricks is to determine whether modular assembly can yield a complete PyTorch code that fully satisfies task requirements. To evaluate this, we conducted 400 independent generation experiments for each of the four mainstream LLM backends across two distinct task orientations, accumulating a total of 1,600 sampling instances. **Figure 3** reports Validity Rate, Pass@1, and CSR under both task orientations. DeepSeek, Qwen, and GPT reach 100% Validity Rate, while LLaMA-3.1 shows minor formatting failures. All four backends maintain Pass@1 above 95%, with the remaining failures mainly caused by runtime tensor-shape mismatches. CSR shows larger backend variation. DeepSeek reaches 100% CSR on accuracy-oriented tasks, compared with 99% for GPT and 97.5% for LLaMA. Efficiency-oriented tasks retain high ValidR and Pass@1 but show lower CSR, indicating that satisfying task preferences remains more backend-sensitive than basic executability.

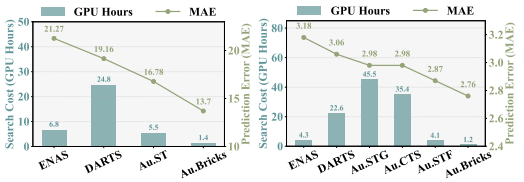


(A) Comparison of generation quality metrics in accuracy-oriented tasks. **(B) Comparison of generation quality metrics in efficiency-oriented tasks.**

Figure 3: Comparison of code generation quality under varying task constraints.

4.4 Comparison of Search Efficiency

We compare GPU-hour costs and forecasting errors on TaxiBJ and METR-LA under the adaptation protocols in **Appendix H**. As shown in **Figure 4**, AutoBricks requires 1.2 to 1.4 GPU hours, compared with up to 45.5 GPU hours for the evaluated baselines. The corresponding cost ratios are roughly 18× and 38× versus DARTS and AutoSTG. **Table 8** further examines operator-pool and search-guidance effects using AutoBricks-9 and Random-38.



(a) TaxiBJ (grid). (b) METR-LA (graph).

Figure 4: Comparison of search efficiency across different spatial topologies.

4.5 Ablation Studies

We use 20 generation samples per variant and task. We compare AutoBricks against four ablations, namely without retrieval (**w/o R.**), without negative constraints (**w/o N.C.**), without the feedback loop (**w/o F.L.**), and without the brick registry (**w/o B.R.**). N/A denotes no executable model. For low-Pass@1 variants, MAE and parameter counts are computed only over executable samples. **Figure 5** shows three consistent findings. The brick registry is the dominant factor for executability: removing it collapses Pass@1, with Qwen on grid tasks dropping from 100% to 5% due to interface hallucinations. Retrieval is the dominant factor for accuracy: w/o R. preserves Pass@1 at 100%

but inflates Qwen's grid MAE from 13.77 to 23.65. The feedback loop and negative constraints contribute smaller but complementary gains, with the former correcting residual generation errors and the latter tightening the boundary against infeasible combinations. The full framework attains near 100% executability and the lowest MAE across the evaluated ablation settings.

4.6 Cost Analysis

Table 4 reports only LLM-side memory, inference latency, and API cost per search round. Candidate-model training is excluded and dominates the end-to-end GPU-hour cost reported in **Figure 4**. Qwen2.5 provides 0.84–0.90 s local inference, while DeepSeek incurs an API cost of $0.00154 per round.

Table 4: Cost analysis across backend LLMs and deployment strategies.

Backend LLM	**LLaMA3.1**		**Qwen2.5**	
Costs	**Mem. (Peak)**	**Infer. Time**	**Mem. (Peak)**	**Infer. Time**
Grid Tasks	55601MB	3.39s	52098MB	0.84s
Graph Tasks	55612MB	3.57s	52110MB	0.90s
Backend LLM	**GPT5-mini**		**DeepSeek**	
Costs	**API Cost**	**Infer. Time**	**API Cost**	**Infer. Time**
Grid Tasks	0.003$	2.15s	0.00154$	2.96s
Graph Tasks	0.003$	2.24s	0.00154$	3.12s

5 Related Work

Spatio-Temporal Forecasting. Beyond classical statistics such as HA (Smith & Demetsky, 1997), and ARIMA (Box et al., 2015), modern ST forecasting splits into grid-based and graph-based deep models. On the grid side, CNN-RNN hybrids ConvLSTM (Shi et al., 2015) and DMVST-Net (Yao et al., 2018), residual designs ST-ResNet (Zhang et al., 2017), DeepSTN+ (Lin et al., 2019), ACFM (Liu et al., 2018b), PCRN (Zonoozi et al., 2018), and the flow gated STDN (Yao et al., 2019) have been progressively extended by Transformer and prompt learning variants STGSP (Zhao et al., 2022), MC-STL (Zhang et al., 2023a), PromptST (Zhang et al., 2023b), and the masked pretrained UniST (Yuan et al., 2024). On the graph side, STGCN (Yu et al., 2018), DCRNN (Li et al., 2018), ASTGCN (Guo et al., 2019), STSGCN (Song et al., 2020), and STAEformer (Liu et al., 2023a) provide representative forecasting architectures, while GWNet (Wu et al., 2019), MTGNN (Wu et al., 2020), AGCRN (Bai et al., 2020), and GTS (Shang et al., 2021) learn adaptive graphs for missing topologies. STGODE (Fang et al., 2021) and D^{2}STGNN (Shao et al., 2022c) model continuous or decoupled dynamics, while DGCRN (Li et al., 2023) generates dynamic discrete topologies and StemGNN (Cao et al., 2020) operates in the spectral domain. Complementary approaches include STNorm (Deng et al., 2021), STID (Shao et al., 2022a), STEP (Shao et al., 2022b), TimeMixer (Wang et al., 2024), and MegaCRN (Jiang et al., 2023).

NAS and LLM Code Generation. General NAS methods such as ENAS and DARTS (Pham et al., 2018; Liu et al., 2018a) motivate ST-specific search in AutoST, AutoSTG, AutoCTS, and AutoSTF (Li et al., 2020; Pan et al., 2021; Wu et al., 2021; Lyu et al., 2025). LLM-driven architecture search includes EvoPrompting, LLMatic, LM-Searcher, and LLM4GNAS (Chen et al., 2023; Nasir et al., 2024; Hu et al., 2025; Gao et al., 2025), with recent work further exploring LLM-guided architecture search for spatio-temporal forecasting and lightweight ST architectures (Xue et al., 2025; Yoon et al., 2025). DS-Agent reuses prior solutions through case-based reasoning (Guo et al., 2024), while Reflexion and TextGrad use language feedback to improve subsequent behavior or system components (Shinn et al., 2023; Yuksekgonul et al., 2025). AutoBricks specializes these ideas for ST architecture design through standardized operator interfaces, positive and negative search memories, and feedback from execution checks and forecasting validation.

6 Conclusion and Future Work

We present AutoBricks, a framework for retrieval-augmented ST architecture search over a standardized brick registry, with a frozen LLM proposing operator compositions. Dual memory retrieves successful configurations and failure records, while execution diagnostics and forecasting validation inform reflective prompt updates that guide subsequent proposals without changing LLM parameters. Experiments across 20 ST datasets evaluate predictive accuracy, code executability, and search efficiency. On PEMS03, the DeepSeek backend reduces MAE by 15.1% relative to AutoSTF, while search-cost comparisons show approximately 3.4- to 38-fold reductions in GPU hours relative to the evaluated NAS baselines. Future directions include enriching the brick registry with multimodal signals, such as maps and metadata, to capture richer spatial context, and developing structured partitioning strategies for scalable model assembly and citywide forecasting.

AI USE STATEMENT

LLMs are part of our method and are used for candidate architecture generation and feedback analysis; their roles and configurations are described in the methodology and experimental setup. We also used LLMs for language polishing and grammar checking. We take full responsibility for the final content of this paper.

ETHICS STATEMENT

AutoBricks aims to lower the barrier to developing spatio-temporal forecasting systems, helping domain practitioners build predictive models for traffic management, environmental monitoring, and public safety. Deployment should respect data licenses and privacy requirements, assess performance disparities across regions, and account for risks arising from prediction errors. Human oversight should be maintained in consequential decision-making.

REPRODUCIBILITY STATEMENT

The methodology describes the brick registry, retrieval-guided assembly, and feedback updates. The experimental setup and appendices document data preprocessing, data splits, baseline adaptation, training configurations, and hyperparameters. Code and demo are available at `https://anonymous.4open.science/r/AutoBricks/` to facilitate inspection of the implementation and experimental workflow.

REFERENCES

Vibhor Agarwal, Yulong Pei, Salwa Alamir, and Xiaomo Liu. Codemirage: Hallucinations in code generated by large language models. *arXiv preprint arXiv:2408.08333*, 2024.

Arsam Aryandoust, Anthony Patt, and Stefan Pfenninger. Enhanced spatio-temporal electric load forecasts using less data with active deep learning. *Nature Machine Intelligence*, 4(11):977–991, 2022.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in neural information processing systems*, 33:17804–17815, 2020.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. *Advances in neural information processing systems*, 36:7787–7817, 2023.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, pp. 6621–6642. PMLR, 2024.

Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Jinliang Deng, Xiusi Chen, Renhe Jiang, Xuan Song, and Ivor W Tsang. St-norm: Spatial and temporal normalization for multi-variate time series forecasting. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 269–278, 2021.

Zheng Fang, Qingqing Long, Guojie Song, and Kunqing Xie. Spatial-temporal graph ode networks for traffic flow forecasting. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 364–373, 2021.

Hamed Farahmand, Yuanchang Xu, and Ali Mostafavi. A spatial–temporal graph deep learning model for urban flood nowcasting leveraging heterogeneous community features. *Scientific Reports*, 13 (1):6768, 2023.

Yang Gao, Hong Yang, Yizhi Chen, Junxian Wu, Peng Zhang, and Haishuai Wang. Llm4gnas: A large language model based toolkit for graph neural architecture search. *arXiv preprint arXiv:2502.10459*, 2025.

Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 922–929, 2019.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*, 2024.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Yuxuan Hu, Jihao Liu, Ke Wang, Jinliang Zheng, Weikang Shi, Manyuan Zhang, Qi Dou, Rui Liu, Aojun Zhou, and Hongsheng Li. Lm-searcher: Cross-domain neural architecture search with llms via unified numerical encoding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9419–9432, 2025.

Pengcheng Jiang, Siru Ouyang, Yizhu Jiao, Ming Zhong, Runchu Tian, and Jiawei Han. Retrieval and structuring augmented generation with large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6032–6042, 2025.

Renhe Jiang, Du Yin, Zhaonan Wang, Yizhuo Wang, Jiewen Deng, Hangchen Liu, Zekun Cai, Jinliang Deng, Xuan Song, and Ryosuke Shibasaki. Dl-traff: Survey and benchmark of deep learning models for urban traffic prediction. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 4515–4525, 2021.

Renhe Jiang, Zhaonan Wang, Jiawei Yong, Puneet Jeph, Quanjun Chen, Yasumasa Kobayashi, Xuan Song, Shintaro Fukushima, and Toyotaro Suzumura. Spatio-temporal meta-graph learning for traffic forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 8078–8086, 2023.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Fan Yang, Funing Sun, Depeng Jin, and Yong Li. Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. *ACM Transactions on Knowledge Discovery from Data*, 17(1):1–21, 2023.

Shuhao Li, Yue Cui, Libin Li, Weidong Yang, Fan Zhang, and Xiaofang Zhou. St-abc: Spatio-temporal attention-based convolutional network for multi-scale lane-level traffic prediction. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1185–1198. IEEE, 2024.

Shuhao Li, Yue Cui, Jingyi Xu, Libin Li, Lingkai Meng, Weidong Yang, Fan Zhang, and Xiaofang Zhou. Unifying lane-level traffic prediction from a graph structural perspective: Benchmark and baseline. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. Autost: Efficient neural architecture search for spatio-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 794–802, 2020.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

Ziqian Lin, Jie Feng, Ziyang Lu, Yong Li, and Depeng Jin. Deepstn+: Context-aware spatial-temporal neural network for crowd flow prediction in metropolis. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 1020–1027, 2019.

Hangchen Liu, Zheng Dong, Renhe Jiang, Jiewen Deng, Jinliang Deng, Quanjun Chen, and Xuan Song. Spatio-temporal adaptive embedding makes vanilla transformer sota for traffic forecasting. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pp. 4125–4129, 2023a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Lingbo Liu, Ruimao Zhang, Jiefeng Peng, Guanbin Li, Bowen Du, and Liang Lin. Attentive crowd flow machines. In *Proceedings of the 26th ACM international conference on Multimedia*, pp. 1553–1561, 2018b.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Xu Liu, Yutong Xia, Yuxuan Liang, Junfeng Hu, Yiwei Wang, Lei Bai, Chao Huang, Zhenguang Liu, Bryan Hooi, and Roger Zimmermann. Largest: A benchmark dataset for large-scale traffic forecasting. *Advances in Neural Information Processing Systems*, 36:75354–75371, 2023b.

Tengfei Lyu, Weijia Zhang, Jinliang Deng, and Hao Liu. Autostf: Decoupled neural architecture search for cost-effective automated spatio-temporal forecasting. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 985–996, 2025.

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. Llmatic: neural architecture search via large language models and quality diversity optimization. In *proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Zheyi Pan, Songyu Ke, Xiaodu Yang, Yuxuan Liang, Yong Yu, Junbo Zhang, and Yu Zheng. Autostg: Neural architecture search for predictions of spatio-temporal graph*. In *Proceedings of the Web Conference 2021*, pp. 1846–1855, 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S Jensen, Zhenli Sheng, et al. Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods. *Proceedings of the VLDB Endowment*, 17(9):2363–2377, 2024.

Feras Saad, Jacob Burnim, Colin Carroll, Brian Patton, Urs Köster, Rif A. Saurous, and Matthew Hoffman. Scalable spatiotemporal prediction with bayesian neural fields. *Nature Communications*, 15(1):7942, 2024.

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

Chao Shang, Jie Chen, and Jinbo Bi. Discrete graph structure learning for forecasting multiple time series. In *International Conference on Learning Representations*, 2021.

Zezhi Shao, Zhao Zhang, Fei Wang, Wei Wei, and Yongjun Xu. Spatial-temporal identity: A simple yet effective baseline for multivariate time series forecasting. In *Proceedings of the 31st ACM international conference on information & knowledge management*, pp. 4454–4458, 2022a.

Zezhi Shao, Zhao Zhang, Fei Wang, and Yongjun Xu. Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1567–1577, 2022b.

Zezhi Shao, Zhao Zhang, Wei Wei, Fei Wang, Yongjun Xu, Xin Cao, and Christian S Jensen. Decoupled dynamic spatial-temporal graph neural network for traffic forecasting. *Proceedings of the VLDB Endowment*, 15(11):2733–2746, 2022c.

Zezhi Shao, Fei Wang, Yongjun Xu, Wei Wei, Chengqing Yu, Zhao Zhang, Di Yao, Tao Sun, Guangyin Jin, Xin Cao, et al. Exploring progress in multivariate time series forecasting: Comprehensive benchmarking and heterogeneity analysis. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Brian L Smith and Michael J Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of transportation engineering*, 123(4):261–266, 1997.

Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 914–921, 2020.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and Jun Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. *arXiv preprint arXiv:2405.14616*, 2024.

Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai Chen, Lei Ma, and Tianyi Zhang. Towards understanding the characteristics of code generation errors made by large language models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2587–2599. IEEE, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xinle Wu, Dalin Zhang, Chenjuan Guo, Chaoyang He, Bin Yang, and Christian S Jensen. Autocts: Automated correlated time series forecasting. *Proceedings of the VLDB Endowment*, 15(4): 971–983, 2021.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 753–763, 2020.

Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. Spatial-temporal transformer networks for traffic flow forecasting. *arXiv preprint arXiv:2001.02908*, 2020.

Xin Xue, Haoyi Zhou, Tianyu Chen, Shuai Zhang, Yizhou Long, and Jianxin Li. Instructing the architecture search for spatial-temporal sequence forecasting with llm. *arXiv preprint arXiv:2503.17994*, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 5668–5675, 2019.

Heeyong Yoon, Jinhong Jung, Kang-Wook Chon, and Min-Soo Kim. Searchlight: Neural architecture search for lightweight spatio-temporal graph neural networks. *IEEE Access*, 2025.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3634–3640, 2018.

Yuan Yuan, Jingtao Ding, Jie Feng, Depeng Jin, and Yong Li. Unist: A prompt-empowered universal model for urban spatio-temporal prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4095–4106, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Xu Zhang, Yongshun Gong, Xinxin Zhang, Xiaoming Wu, Chengqi Zhang, and Xiangjun Dong. Mask-and contrast-enhanced spatio-temporal learning for urban flow prediction. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pp. 3298–3307, 2023a.

Zijian Zhang, Xiangyu Zhao, Qidong Liu, Chunxu Zhang, Qian Ma, Wanyu Wang, Hongwei Zhao, Yiqi Wang, and Zitao Liu. Promptst: Prompt-enhanced spatio-temporal multi-attribute prediction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 3195–3205, 2023b.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. In *International Conference on Machine Learning*, pp. 61121–61143. PMLR, 2024.

Liang Zhao, Min Gao, and Zongwei Wang. St-gsp: Spatial-temporal global semantic representation learning for urban flow prediction. In *Proceedings of the Fifteenth ACM international conference on web search and data mining*, pp. 1443–1451, 2022.

Ali Zonoozi, Jung-jae Kim, Xiao-Li Li, and Gao Cong. Periodic-crn: A convolutional recurrent model for crowd density prediction with recurring periodic patterns. In *Ijcai*, volume 18, pp. 3732–3738, 2018.

A Statistical Significance Analysis

To assess the stability of the reported results, we repeated the selected AutoBricks configuration and the strongest baseline on 6 representative datasets with 5 random seeds, rerunning the complete search and training procedure for each seed. Paired t-tests are conducted across matched seed runs. On all 3 graph datasets, the selected AutoBricks configurations outperform AutoSTF at the $p < 0.01$ level. The largest improvement occurs on PEMS03, where AutoBricks-D achieves an MAE of 10.48 ± 0.45, a 15.3% reduction over AutoSTF's 12.37 ± 0.58, with values reported as mean $\pm$ standard deviation. Similar gains are observed on PEMS08 (8.6%, 11.21 ± 0.28 vs. 12.26 ± 0.32) and PEMS07 (12.2%, 14.68 ± 0.60 vs. 16.72 ± 0.65), showing that the observed gains remain consistent across the evaluated random seeds. On the 3 grid datasets, the gap between AutoBricks-L and UniST narrows to 1.1-1.7%, yet paired tests still reject the null hypothesis at $p < 0.05$. AutoBricks-L's standard deviations remain within 0.03 across all three datasets, indicating that the advantage is modest but consistently reproducible.

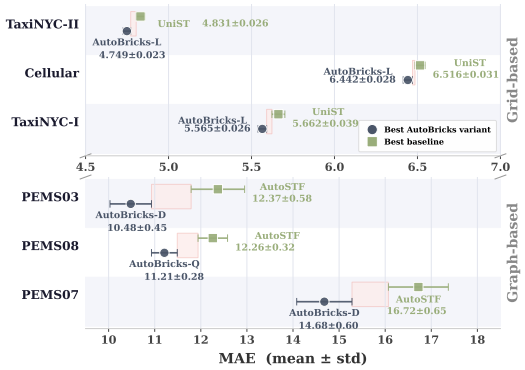


Figure 6: Statistical significance of AutoBricks performance gains across six datasets.

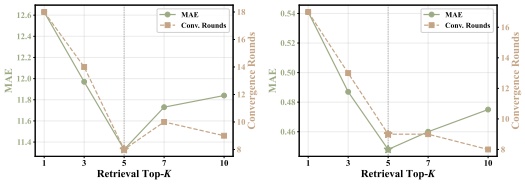
The strongest backend varies across datasets and topologies. The CSR analysis in **Section 4.3** further shows backend-dependent differences in satisfying task constraints, indicating that backend selection remains task-dependent.

B Hyperparameter Sensitivity Analysis

To evaluate the robustness to core hyperparameters, we sweep four key hyperparameters across different backend and data topology combinations, fixing all others at their default values. For analysis, the convergence round is the first round that reaches the final best MAE within the 20-round search budget.

B.1 Number of retrieval anchors Top-K

We sweep $K \in \{1, 3, 5, 7, 10\}$ on Qwen+PEMS08 and GPT+TrafficNJ. Both settings exhibit a consistent decrease-then-increase trend, with the optimal point at $K = 5$. On Qwen+PEMS08, MAE drops from 12.63 at $K = 1$ to 11.33 at $K = 5$ (10.3% reduction); on GPT+TrafficNJ, from 0.541 to 0.448 (17.2%). Convergence rounds improve correspondingly, nearly halving from 18/17 rounds at $K = 1$ to 8/9 rounds at $K = 5$. Beyond $K = 5$, MAE slightly increases (11.84 on PEMS08 at $K = 10$), indicating that excessive anchors introduce contextual noise that offsets information gain. Notably, convergence rounds saturate at $K \geq 7$ (9–10 rounds), suggesting that additional anchors accelerate initial localization but offer no benefit for fine-grained tuning.

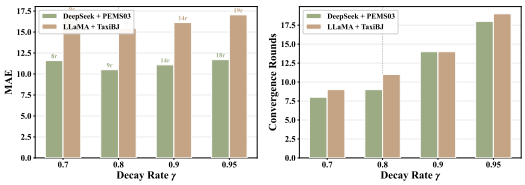


(a) Qwen+PEMS08. (b) GPT+TrafficNJ.

Figure 7: Effect of retrieval anchor count K on MAE and convergence rounds.

B.2 Exploration decay rate γ

We sweep $\gamma \in \{0.7, 0.8, 0.9, 0.95\}$ on DeepSeek+PEMS03 and LLaMA+TaxiBJ. Both settings show a U-shaped curve with the optimal point consistently at $\gamma = 0.8$. On DeepSeek+PEMS03, MAE at $\gamma = 0.8$ is 10.51, while $\gamma = 0.7$ and $\gamma = 0.95$ degrade to 11.60 (+10.4%) and 11.71 (+11.4%) respectively, through different mechanisms: the former traps the search in the initial anchor neighborhood due to insufficient exploration, while the latter exhausts over 60% of the budget on random sampling within 20 rounds. LLaMA+TaxiBJ corroborates this pattern, with $\gamma = 0.8$ yielding MAE 15.45 and symmetric degradation at both extremes (17.18 and 17.05). Convergence rounds increase monotonically with γ (from 8 to 18 rounds on DeepSeek), further confirming that overly slow decay wastes the search budget.

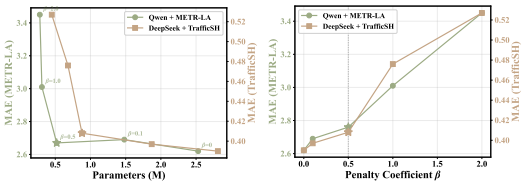


(a) Comparison on MAE. (b) Convergence Rounds.

Figure 8: Effect of exploration decay rate γ on MAE and convergence rounds.

B.3 Reward trade-off coefficient β

We sweep $\beta \in \{0, 0.1, 0.5, 1.0, 2.0\}$ on Qwen+METR-LA and DeepSeek+TrafficSH. Both settings exhibit a nonlinear accuracy-efficiency trade-off. On Qwen+METR-LA, increasing β from 0 to 0.5 compresses parameter count from 2.53M to 0.52M (4.9× reduction) with a 5.3% MAE increase (2.62 to 2.76), whereas further increasing β to 2.0 reduces parameters to 0.28M with smaller absolute savings but raises MAE to 3.45 (+31.7% relative to $\beta = 0$). DeepSeek+TrafficSH exhibits the same nonlinear pattern, where the accuracy cost at $\beta = 2.0$ far exceeds the marginal gain in parameter reduction. For these two evaluated settings, $\beta = 0.5$ lies near the observed accuracy-efficiency trade-off knee. Because MAE remains task-scaled, this result does not imply an identical trade-off across datasets.

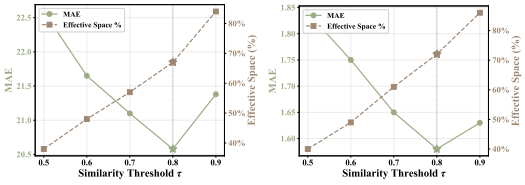


(a) Parameters vs. MAE. (b) MAE vs. β.

Figure 9: Accuracy–efficiency trade-off under varying reward coefficient β

B.4 Negative memory similarity threshold τ

We sweep $\tau \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ on GPT+PEMS07 and LLaMA+Crowd. Both settings show a U-shaped MAE curve with the optimal point at $\tau = 0.8$. On GPT+PEMS07, $\tau = 0.5$ reduces the estimated effective-space ratio to 38% and degrades MAE to 22.59 (+9.8% over the optimal 20.58), while $\tau = 0.9$ retains 84% and yields MAE 21.38 (+3.9%). LLaMA+Crowd shows a flatter U-shape with more sensitivity to over-pruning ($\tau = 0.5$: MAE 1.84, +16.5%; $\tau = 0.9$: 1.63, +3.2%), consistent with the smaller search space of this dataset. Across both settings, $\tau = 0.8$ yields the lowest MAE in the evaluated sweep while retaining a moderate effective-space ratio.



(a) GPT+PEMS07. (b) LLaMA+Crowd.

Figure 10: Effect of negative memory threshold τ on MAE and effective search space.

Table 5: Cross-task memory transfer results comparing cold-start (default main-experiment configuration) and warm-start. All experiments use DeepSeek with a 20-round budget.

Target Dataset	Conv. Rounds		Speedup Ratio	MAE	
	Cold	Warm		Cold	Warm
PEMS07	11	6	1.83x	14.88	14.58
TaxiNYC-I	10	6	1.67x	5.865	5.695
BikeNYC-II	9	5	1.80x	2.353	2.284
TrafficSH	9	5	1.80x	0.408	0.394
Average	9.75	5.5	1.78x	**Reduction**	2.82%

C Search Convergence Dynamics

Figure 11 shows a representative 20-round DeepSeek+PEMS03 search trajectory, tracking the current best MAE, the per-round Pass@1 over 20 independent generation samples, and the estimated effective-space ratio.

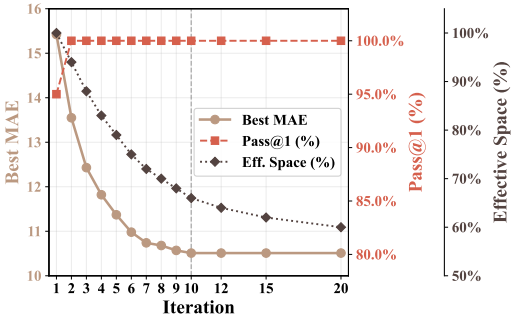


Figure 11: Convergence dynamics of AutoBricks over 20 search rounds.

The best observed MAE decreases from 15.42 in round 1 to 10.98 in round 6 and reaches 10.51 in round 10, with no further improvement reported through round 20. This trajectory shows rapid early improvement followed by an empirical plateau. The per-round Pass@1, computed from 20 generation samples, increases from 95% in round 1 to 100% from round 2 onward. The initial failures mainly arise from operator-specific shape constraints, while no execution failures are observed in the sampled generations from round 2 onward. The reported effective-space ratio decreases from 100% initially to 66% by round 10 and 60% by round 20, with the largest changes occurring early in the search. This decreasing trajectory is an empirical observation under the evaluated retrieval sequence. Under the top-K filtering rule, refreshing negative records can also restore previously excluded regions, as analyzed in **Appendix E**. The MAE plateau around round 10 coincides with slower changes in the reported space ratio, but this association does not establish a causal mechanism or convergence to a globally optimal architecture.

D Cross-Task Memory Transfer Analysis

The main experiments in **Section 4** adopt a cold-start configuration, the dual memory $\mathcal{M}^+ \cup \mathcal{M}^-$ is initialized as empty for each target task and accumulated on the fly during the 20-round search, so the main experiments do not use search memories transferred from other tasks. Building on this default, we further examine whether prior search experience can accelerate convergence by introducing a *warm-start* configuration that loads $\mathcal{M}^+$ and $\mathcal{M}^-$ pre-accumulated from a fixed 10-task source pool spanning both grid and graph topologies. Both configurations use DeepSeek with the same 20-round budget on four representative target tasks. As shown in **Table 5**, warm-start reduces the average number of convergence rounds from 9.75 to 5.5, a mean per-task speedup ratio of 1.78, with the largest gains in intra-domain transfer. PEMS07 converges in 6 vs. 11 rounds by reusing diffusion convolution anchors from PEMS03/04/08, and the grid-speed task TrafficSH drops from 9 to 5 rounds via traffic-speed memories from TrafficNJ/CD/HZ. Final-MAE improvement averages 2.82% (computed per dataset to avoid scale bias), with TrafficSH achieving the largest relative gain of 3.43% and PEMS07 the smallest at 2.02%. Across these four tasks, warm-start reduces the rounds required to reach the reported MAE plateau and yields modest improvements in final MAE. The main results in **Tables 2** and **3** use cold-start to evaluate search without transferred task memories.

Both configurations reach a reported Pass@1 of 100% within one or two rounds, with averages of 1.3 rounds for cold-start and 1.0 for warm-start. This indicates high observed executability under

both initializations. On the four evaluated tasks, warm-start reaches the reported MAE plateau in five to six rounds, compared with nine to eleven rounds for cold-start, supporting the usefulness of transferred search experience.

E Formalization and Analysis of the In-Context Search Process

We formalize AutoBricks as sequential architecture search driven by a frozen LLM. We specify the reward, cost, and similarity measures, describe the interaction between proposals, evaluation, memory, and prompt updates, and analyze how these updates can affect expected reward and candidate exclusion. We also characterize the exploration frequency induced by the exponential mixture schedule.

E.1 Concrete Definitions of Reward, Cost, and Similarity

Reward and MAE. The reward in **Eq. 9** uses validation MAE on the evaluation scale adopted for each task. We keep this scale and the coefficients $\alpha = 1.0$ and $\beta = 0.5$ fixed when comparing candidates within a task. Since inverse MAE is scale-dependent, these coefficients do not imply an identical accuracy–efficiency trade-off across tasks.

Cost normalization. We normalize parameter count using a fixed reference value and clip the result to $[0, 1]$,

$$\mathrm{Cost}(f_{\mathcal{A}_t}) = \min\left(\frac{|\theta_{\mathcal{A}_t}|}{P_{\max}}, 1\right). \tag{11}$$

Here $|\theta_{\mathcal{A}_t}|$ is the assembled model's parameter count, and $P_{\max} = 10\text{M}$ is the normalization reference used in the experiments. Hard resource constraints are checked separately, while this bounded parameter-size penalty guides the search.

Architectural similarity. The similarity function in **Eq. 4** is the Jaccard index between sets of brick names. For a nonempty architecture Φ, let $T(\Phi)$ denote the set of its distinct brick names. We define

$$\mathrm{Sim}(\Phi_1, \Phi_2) = \frac{|T(\Phi_1) \cap T(\Phi_2)|}{|T(\Phi_1) \cup T(\Phi_2)|}. \tag{12}$$

With the default threshold $\tau = 0.8$, candidates exceeding this similarity to a retrieved structural negative record are excluded before training. For compact pipelines with at most four distinct bricks, this strict threshold typically triggers only when the brick-name sets are identical. Because the measure ignores ordering, connectivity, and hyperparameters, exclusion remains a heuristic rather than a proof of infeasibility.

E.2 Sequential Search Formulation

AutoBricks maintains an evolving prompt and experience memories while using a frozen LLM to propose architectures. We describe this sequential process without assuming that the prompt alone is a Markov state.

Definition E.1 (Search History and Prompt)**.** *Let $\mathcal{H}_t$ denote the search history available before round t, including previous proposals, evaluation outcomes, and prompt updates. The LLM receives a bounded context S_t that contains the task description $\mathcal{T}$, the retrieved records $\mathcal{R}_t^+$ and $\mathcal{R}_t^-$, and retained feedback. The evolving memories $\mathcal{M}_t^+$ and $\mathcal{M}_t^-$, retained evaluation records, and round index also influence subsequent retrieval, candidate selection, and exploration. Thus, the prompt summarizes information available to the LLM but need not determine the entire search process.*

Definition E.2 (Action Space)**.** *The configuration space Ω contains assembly plans specifying brick instances from $\mathcal{B}$, their DAG connections, fusion strategies at multi-parent nodes, and operator hyperparameters. Where finite sums or cardinalities are used below, we assume that the search protocol bounds the number of brick instances and restricts hyperparameters to finite sets. A finite registry alone does not establish these conditions, and interface standardization does not guarantee that every composition passes execution checks.*

Definition E.3 (Evaluation and Context Update)**.** *Candidates passing the similarity filter undergo instantiation and a trial forward pass. Candidates that fail this stage skip full training; those that pass*

proceed to training and validation. Let $R_t^{\text{best}} = \max\{R(\mathcal{A}_j) \mid j < t,\ \mathcal{A}_j \text{ completes evaluation}\}$, *with* $R_t^{\text{best}} = -\infty$ *before the first completed evaluation. A completed candidate enters* $\mathcal{M}^+$ *if* $R(\mathcal{A}_t) > R_t^{\text{best}}$*; failed or non-improving evaluations are retained in* $\mathcal{M}^-$ *with their failure types. Retrieval then refreshes* $\mathcal{R}_{t+1}^+$ *and* $\mathcal{R}_{t+1}^-$*, and the resulting feedback is summarized into the next prompt.*

Definition E.4 (Evaluation Reward)**.** *For candidates entering execution evaluation, we distinguish failed evaluations from completed training and validation,*

$$R(\mathcal{A}_t) = \begin{cases} -\rho, & \text{if evaluation fails}, \\ \dfrac{\alpha}{\mathrm{MAE}(f_{\mathcal{A}_t}, \mathcal{D}_{\text{val}}) + \eta} - \beta\,\mathrm{Cost}(f_{\mathcal{A}_t}), & \text{if training and validation complete}. \end{cases} \tag{13}$$

Here $\rho > 0$ *is the failure penalty. Failed evaluations provide diagnostic feedback without a validation MAE. Rejection by the similarity filter is a separate event preceding execution evaluation.*

Definition E.5 (Mixture Proposal)**.** *Before filtering, candidates are generated from*

$$\pi_{\text{mix}}(\mathcal{A} \mid S_t) = \epsilon_t \pi_{\text{rand}}(\mathcal{A}) + (1 - \epsilon_t)\pi_{\text{LLM}}(\mathcal{A} \mid S_t), \tag{14}$$

where π_{rand} *is the random configuration sampler,* π_{LLM} *is the frozen LLM conditioned on* S_t*, and* $\epsilon_t = \epsilon_{\text{start}}\gamma^t$ *with* $0 < \gamma < 1$*. Uniform sampling over complete architectures is not required.*

For a proposal drawn from this mixture, conditioning on passing the common exclusion filter gives

$$q_t^{\text{mix}}(\mathcal{A} \mid S_t) = \frac{\pi_{\text{mix}}(\mathcal{A} \mid S_t)\mathbb{I}(\mathcal{A} \notin \mathcal{C}_t^-)}{\sum_{\mathcal{A}' \in \Omega} \pi_{\text{mix}}(\mathcal{A}' \mid S_t)\mathbb{I}(\mathcal{A}' \notin \mathcal{C}_t^-)}. \tag{15}$$

This distribution is defined when the denominator is positive. The q_t *in the main paper conditions the LLM branch alone on passing the filter, whereas* q_t^{mix} *conditions the full mixture. Filtering can change the relative proportions of random and LLM-generated candidates.*

The LLM parameters remain fixed. Updating the context changes the conditional proposal distribution, without implying that its expected reward necessarily increases.

E.3 Reflective Prompt Updates

AutoBricks uses textual gradients to denote natural-language guidance derived from execution diagnostics and validation feedback. The reflection operator produces

$$u_t = \mathrm{Reflect}(\mathcal{F}_t), \tag{16}$$

and the next context is constructed as

$$S_{t+1} = \mathrm{Update}\big(S_t, u_t, \mathcal{R}_{t+1}^+, \mathcal{R}_{t+1}^-\big). \tag{17}$$

For example, an OOM diagnostic can motivate lighter operator combinations that pass the similarity filter. Prior work on in-context learning and chain-of-thought prompting motivates conditioning generation on examples and intermediate reasoning (Brown et al., 2020; Wei et al., 2022). Reflection changes the context supplied to the frozen LLM. The following proposition expresses the resulting change in expected reward through the reward averages of gained and lost probability mass.

Proposition E.1 (Conditional Change in Expected Reward)**.** *For a fixed task and evaluation protocol, let* $\mu(\mathcal{A})$ *denote the expected reward of architecture* $\mathcal{A}$ *over evaluation randomness. Let* p *and* p' *be normalized architecture distributions over the same finite* Ω *before and after a context update, and define*

$$\delta_+(\mathcal{A}) = \max\{p'(\mathcal{A}) - p(\mathcal{A}), 0\}, \qquad \delta_-(\mathcal{A}) = \max\{p(\mathcal{A}) - p'(\mathcal{A}), 0\}. \tag{18}$$

The total gained and lost probability masses are equal, so

$$m = \sum_{\mathcal{A} \in \Omega} \delta_+(\mathcal{A}) = \sum_{\mathcal{A} \in \Omega} \delta_-(\mathcal{A}). \tag{19}$$

For $m > 0$*, define the normalized distributions* $\nu_+ = \delta_+/m$ *and* $\nu_- = \delta_-/m$*. With* $J(p) = \sum_{\mathcal{A} \in \Omega} p(\mathcal{A})\mu(\mathcal{A})$*, we have*

$$J(p') - J(p) = m\left(\mathbb{E}_{\nu_+}[\mu(\mathcal{A})] - \mathbb{E}_{\nu_-}[\mu(\mathcal{A})]\right). \tag{20}$$

Therefore, expected reward does not decrease if the gained mass has mean reward at least as high as the lost mass. When $m = 0$*, the two distributions coincide.*

Proof. Since $p' - p = \delta_+ - \delta_-$, expanding $J(p') - J(p)$ gives the difference between the reward-weighted gained and lost masses. Substituting $\delta_+ = m\nu_+$ and $\delta_- = m\nu_-$ yields the identity. □

This identity states a condition for improvement. Reducing the probability of one failed architecture alone does not establish this condition, because other architectures may also lose probability mass. AutoBricks does not guarantee that every reflective update improves expected reward.

E.4 Retrieved Exclusions and the Effective Search Space

The retrieved structural negative records define the exclusion set $\mathcal{C}_t^-$ in **Eq. 4**. The effective configuration space is

$$\Omega_t^{\text{eff}} = \Omega \setminus \mathcal{C}_t^-. \tag{21}$$

Because exclusion is induced only by the currently retrieved structural records, refreshing retrieval can both add and remove exclusion regions.

Proposition E.2 (Effect of Updating Retrieved Exclusions)**.** *For finite Ω, write $\mathcal{C}_t^- = B \cup U$ and $\mathcal{C}_{t+1}^- = B \cup V$, where B is the union of regions induced by retained records, U the union induced by removed records, and V the union induced by added records. Then*

$$|\Omega_{t+1}^{\text{eff}}| - |\Omega_t^{\text{eff}}| = |U \setminus B| - |V \setminus B|. \tag{22}$$

Consequently, the effective space does not expand when the newly excluded coverage outside B is at least as large as the coverage lost outside B.

Proof. The identities $|B \cup U| = |B| + |U \setminus B|$ and $|B \cup V| = |B| + |V \setminus B|$, together with $|\Omega_t^{\text{eff}}| = |\Omega| - |\mathcal{C}_t^-|$, give the result. □

TF-IDF relevance does not guarantee this coverage condition. Retaining all previous exclusion regions would ensure non-expansion, but this is a different filtering rule from using only the currently retrieved records. Under the stated top-K rule, the effective space may expand or contract between rounds. Membership in this space indicates that a candidate passes the similarity filter, not that it is executable or accurate. Figures 10 and 11 report the fraction $|\widetilde{\Omega} \setminus \mathcal{C}_t^-|/|\widetilde{\Omega}|$ of a fixed task-specific reference pool $\widetilde{\Omega}$ that remains after filtering.

E.5 Exploration Frequency under the Exponential Schedule

Classical bandit regret guarantees depend on the reward assumptions and action-selection rule (Auer et al., 2002). Here, we characterize the exploration frequency implied by the AutoBricks schedule. For a fixed horizon of n rounds indexed by $t = 0, \ldots, n-1$, let I_t^{rand} indicate whether the initial mixture choice in round t selects the random branch, and let $N_{\text{rand}}(n) = \sum_{t=0}^{n-1} I_t^{\text{rand}}$. Since $\mathbb{E}[I_t^{\text{rand}}] = \epsilon_{\text{start}}\gamma^t$, linearity of expectation gives

$$\mathbb{E}[N_{\text{rand}}(n)] = \epsilon_{\text{start}} \frac{1-\gamma^n}{1-\gamma} \leq \frac{\epsilon_{\text{start}}}{1-\gamma}. \tag{23}$$

For $\epsilon_{\text{start}} = 1.0$ and $\gamma = 0.8$, this expectation is bounded by 5 and is approximately 4.94 over 20 rounds. The calculation counts initial branch choices, excluding additional proposals caused by rejection or repair. It does not bound training time or establish convergence to an optimal architecture. Such a guarantee would additionally require control of the LLM proposal quality and the effects of filtering. The observed search trajectories and GPU costs therefore remain empirical results.

F Expressiveness and Extensibility of the Standardized Brick Registry

Following the search-process analysis in **Appendix E**, we examine the architectural scope of the 38-brick registry through design-axis coverage, conditions for baseline reconstruction, and compositions beyond the source models.

Table 6: Coverage of the AutoBricks brick registry across ST design axes.

Axis	Category	Bricks
Spatial	Grid convolution	`CNN, GateCNN, ResConv, ResPlusConv, LocalCNN`
	Grid residual / attention	`SpatialResNet, SpatialTransformer, ViTGCN`
	Graph spectral / diffusion	`GCN, ChebGCN, DiffusionGCN`
	Graph adaptive / learned	`AdaptiveGCN, GumbelGraph, MixHopGCN`
	Graph attention	`GAT, SpatialAttention`
	Embedding / prompt	`SpatialEmbedding, TimeDisConv, SFEncoder`
Temporal	Recurrent	`LSTM (Graph/Grid), GRU, ConvLSTM, PyramidalConvGRU`
	Attention-augmented recurrent	`AttentionLSTM, AttentionGRU, AttentionConvLSTM`
	Convolutional	`DilatedTCN, DilatedInceptionTCN, GLUTCN, WaveNet`
	Attention / Transformer	`Transformer, BiTransformer, TemporalAttention`
	Residual / MLP / prompt	`TemporalResNet, ResidualMLP, TemporalPrompt, EarlyFusionCNN`

F.1 Design-Axis Coverage

ST architecture design involves spatial inductive biases, temporal modeling mechanisms, and fusion topologies. **Table 6** summarizes the spatial and temporal operator families represented in the registry.

The registry covers multiple spatial and temporal operator families. The unified 4D interface facilitates assembly, while valid combinations still depend on operator-specific shape and topology requirements. This coverage supports architectural variety within the registry but does not establish completeness over these model families.

F.2 Reproducibility of Baseline Architectures

Proposition F.1 (Conditions for Exact Reconstruction)**.** *Consider a baseline represented as a computation DAG. If each computation has a semantically equivalent registry implementation, its wiring can be reproduced, and interface transformations preserve its operations, then the assembled model computes the same function under corresponding parameters.*

Proof. Traverse the DAG in topological order. Corresponding input nodes receive identical tensors. If corresponding parent nodes produce identical outputs, equivalent operations with corresponding parameters produce identical outputs at the next node. Induction establishes equality at the output nodes. □

Proposition F.1 applies only when all required computations and interface-preserving transformations are represented in the registry. Baselines requiring unregistered coupled recurrent operations, multi-window processing, or continuous solvers are represented by standardized variants rather than exact functional reconstructions. In our experiments, variants of the 30 deep baselines matched their reported accuracy within $\pm 2\%$ on datasets with published results.

F.3 Compositions Beyond Source Models

Candidate architectures can differ from the source models through operator choices and DAG connections. Their validity remains subject to tensor compatibility and topology requirements. In particular, reshaping graph nodes into a grid does not by itself preserve spatial neighborhood semantics, so cross-topology combinations require suitable adapters and explicit justification. Across the 20 datasets and four LLM backends, approximately 34% of final architectures were non-isomorphic to any source baseline under typed-DAG comparison, where isomorphism preserves connectivity and brick types while ignoring scalar hyperparameters.

F.4 Extensibility

The registry can be extended with operators that satisfy the invocation interface $\mathbf{Z} = b_i(X, \mathcal{G}; \Phi_i)$ and the required shape and topology conventions. Integration requires registering the operator, checking compatibility, and exposing its description to candidate generation. These changes do not require updating LLM parameters. Once configurations using the new brick are evaluated and

Table 7: Statistics of datasets.

Grid-based Datasets				Grid-based Datasets			
Dataset	**Domains**	**Region**	**Grid Size**	**Dataset**	**Domains**	**Region**	**Grid Size**
TaxiBJ	Taxi GPS	Beijing	32 × 32	**Crowd**	Crowd flow	Nanjing	16 × 20
Cellular	Cellular usage	Nanjing	16 × 20	**BikeNYC-I**	Bike usage	New York	16 × 12
BikeNYC-II	Bike usage	New York	10 × 20	**TaxiNYC-I**	Taxi OD	New York	16 × 12
TaxiNYC-II	Taxi OD	New York	20 × 10	**TrafficNJ**	Traffic speed	Nanjing	32 × 24
TrafficCD	Traffic speed	Chengdu	28 × 26	**TrafficSH**	Traffic speed	Shanghai	28 × 32
TrafficHZ	Traffic speed	Hangzhou	28 × 24	**TrafficJN**	Traffic speed	Jinan	32 × 18
TrafficTJ	Traffic speed	Tianjin	24 × 30	**TrafficZZ**	Traffic speed	Zhengzhou	26 × 26
Graph-based Datasets				**Graph-based Datasets**			
Dataset	**Domains**	**Region**	**# Nodes**	**Dataset**	**Domains**	**Region**	**# Nodes**
METR-LA	Traffic speed	Los Angeles	207	**PEMS-BAY**	Traffic speed	Bay Area	325
PEMS03	Traffic flow	Sacramento	358	**PEMS04**	Traffic flow	Bay Area	307
PEMS07	Traffic flow	Los Angeles	883	**PEMS08**	Traffic flow	Bernardino	170

stored, subsequent retrieval can reuse those records. Existing memories do not automatically contain experience with a newly introduced operator.

G Dataset Description

We evaluate AutoBricks on **20 datasets** spanning two topologies and six urban sensing domains (traffic flow, traffic speed, human mobility, bike sharing, taxi OD, cellular activity). Spatial scales range from 10×20 grids to 883-node graphs, providing a stress test across heterogeneous data structures and forecasting horizons. Statistics are summarized in **Table 7**.

G.1 Grid-Based Datasets

The 14 grid-based datasets are organized over a regular Euclidean lattice and capture three categories of urban dynamics.

Human mobility and demand (7 datasets). *TaxiBJ* records taxi inflow and outflow in Beijing partitioned over a 32×32 grid, sampled at 30-minute intervals over four years; it is one of the most widely used benchmarks for citywide crowd flow prediction. *Crowd* and *Cellular* capture pedestrian flow and cellular signaling activity in Nanjing on a 16×20 grid. Following prior practice (Yuan et al., 2024; Zhang et al., 2023b), the raw values of these two datasets are scaled by 1,000 to facilitate visualization and to keep the loss magnitudes comparable across datasets. *BikeNYC-I/II* and *TaxiNYC-I/II* record bike-sharing demand and taxi origin-destination flows in New York at different spatial resolutions and time periods, capturing both fine-grained short-term dynamics and longer seasonal patterns.

Citywide traffic speed (7 datasets). The *Traffic-XX* family covers seven major Chinese cities (Nanjing, Chengdu, Shanghai, Hangzhou, Jinan, Tianjin, Zhengzhou). Each dataset gridifies the road network into rectangular cells and aggregates GPS-derived vehicle speeds at 5-minute intervals. The diverse city sizes (from 24×30 to 32×24) and traffic patterns provide a stress test for cross-region generalization.

For all grid datasets, each observation tensor takes the shape $X_t \in \mathbb{R}^{H' \times W' \times C}$, where C denotes the number of measured channels. Spatial locality is encoded directly by the Euclidean lattice coordinates, and grid operators recover (H', W') from dataset metadata without consuming $\mathcal{G}$.

G.2 Graph-Based Datasets

The 6 graph-based datasets are non-Euclidean and represent traffic sensors as graph nodes connected by an explicit adjacency matrix $A \in \mathbb{R}^{N \times N}$ derived from road network distances. *METR-LA* and *PEMS-BAY* (Li et al., 2018) are the two canonical traffic speed benchmarks, containing loop-detector readings from highways in Los Angeles County and the San Francisco Bay Area, respectively. The four *PEMS-XX* datasets are public PeMS traffic-flow benchmarks widely used in prior work (Guo et al., 2019; Song et al., 2020), recording traffic flow (vehicles per 5 minutes) across different California districts, with PEMS07 being the largest at 883 sensor nodes. These graph datasets are characterized

by strong spatial dependencies, long-range correlations, and pronounced non-stationarity, making them particularly challenging for long-horizon forecasting.

G.3 Preprocessing and Splits

Across all datasets, we follow the standard preprocessing pipeline established in the DL-Traff benchmark (Jiang et al., 2021) and BasicTS+ (Shao et al., 2024) for fair comparison. Specifically: (i) all features are normalized using Z-score standardization computed on the training set; (ii) missing values, where present, are linearly interpolated along the temporal dimension; (iii) for grid datasets, the train/validation/test split follows a chronological 7:1:2 ratio; for the four PEMS-XX graph datasets, we adopt the conventional 6:2:2 split, while METR-LA and PEMS-BAY follow the 7:1:2 split as in DCRNN (Li et al., 2018). The historical input window T and prediction horizon H are set to $T = H = 6$ for grid tasks and $T = H = 12$ for graph tasks, consistent with the protocols used by the strongest baselines in each category.

H Baselines Details

We compare AutoBricks against a comprehensive suite of 40 baselines spanning three paradigms: (i) human-designed models, (ii) LLM-based code generation, and (iii) automated search frameworks. This section details the baselines and the adaptation protocol used to ensure a fair comparison.

H.1 Human-Designed Models

Statistical baselines. Historical Average (HA) (Smith & Demetsky, 1997) predicts each future timestep as the mean of historical values at the same time-of-day. ARIMA (Box et al., 2015) is a classical autoregressive integrated moving average model fitted independently per spatial unit. These methods provide a non-trivial floor that captures periodicity without learning spatial dependencies.

Grid-based deep models (12 methods). This category reflects the evolution of grid-based ST modeling. ConvLSTM (Shi et al., 2015) integrates convolutional structures into LSTM gates for joint spatio-temporal encoding. ST-ResNet (Zhang et al., 2017) introduces residual learning with closeness, period, and trend branches. DMVST-Net (Yao et al., 2018) and STDN (Yao et al., 2019) couple CNN-LSTM with attention and flow-gating. ACFM (Liu et al., 2018b) uses attentive crowd flow machines for spatial focusing. PCRN (Zonoozi et al., 2018) is a pyramidal convolutional recurrent model for periodic patterns. DeepSTN+ (Lin et al., 2019) injects context priors into deep residual ST networks. STGSP (Zhao et al., 2022) and MC-STL (Zhang et al., 2023a) leverage transformers and mask-contrast pretraining. PromptST (Zhang et al., 2023b) applies prompt learning to multi-attribute ST prediction. PatchTST (Nie et al., 2022) and UniST (Yuan et al., 2024) are pretrained foundation models—the former a patch-based time series transformer, the latter a prompt-empowered universal model trained on multi-city corpora.

Graph-based deep models (18 methods). Graph baselines span the major architectural families. *Graph and attention-based backbones* include STGCN (Yu et al., 2018), DCRNN (Li et al., 2018), ASTGCN (Guo et al., 2019), STSGCN (Song et al., 2020), STAEformer (Liu et al., 2023a), and StemGNN (Cao et al., 2020). *Adaptive graph learning*: GWNet (Wu et al., 2019), MTGNN (Wu et al., 2020), AGCRN (Bai et al., 2020), DGCRN (Li et al., 2023), and GTS (Shang et al., 2021), which learn the adjacency matrix when the topology is missing or noisy. *Continuous and decoupled dynamics*: D^{2}STGNN (Shao et al., 2022c), STGODE (Fang et al., 2021). *Lightweight and efficiency-focused*: STNorm (Deng et al., 2021), STID (Shao et al., 2022a), and TimeMixer (Wang et al., 2024). *Meta-graph learning*: MegaCRN (Jiang et al., 2023). *Pretrained graph foundation*: STEP (Shao et al., 2022b), which pretrains a scalable transformer encoder on long-term series before downstream forecasting.

H.2 LLM-based Code Generation Baselines

Codex. We evaluate prompt-based code generation through Codex using GPT-Astra-6-pro. Codex receives the same task description $\mathcal{T}$ as AutoBricks, including dataset shape, prediction horizon, and resource budget, without the AutoBricks brick registry or retrieved memories. The prompt requests a

runnable PyTorch `nn.Module` definition. We allow Codex to iterate until it produces a runnable model or reaches 10 refinement rounds.

Claude Code. We evaluate Claude Code powered by Claude Fable 5.1. Claude Code uses tool-assisted iterative refinement to debug and improve generated forecasting code, including sandbox execution, error inspection, and code revision. We provide Claude Code with the same task description $\mathcal{T}$ as AutoBricks and allow it to iterate until it produces a runnable model or reaches 10 refinement rounds. This baseline evaluates agentic multi-turn debugging as an alternative workflow for ST model generation.

H.3 Automated Search Frameworks

General NAS. *ENAS* (Pham et al., 2018) accelerates architecture search by sharing parameters within a supernet, controlled by an RNN policy. *DARTS* (Liu et al., 2018a) relaxes the discrete architecture choice into a differentiable mixture and jointly optimizes architecture and weights via bi-level optimization.

ST-specific AutoML. *AutoST* (Li et al., 2020) designs a search space tailored for grid-based ST prediction, combining residual blocks with closeness, period, and trend branches. *AutoSTG* (Pan et al., 2021) extends NAS to spatio-temporal graphs with meta-learned graph operators. *AutoCTS* (Wu et al., 2021) jointly searches operator topology and hyperparameters for correlated time series. *AutoSTF* (Lyu et al., 2025) decouples temporal and spatial search to reduce cost while preserving accuracy and represents the strongest ST NAS baseline in our experiments.

Human-designed baselines. We reuse official hyperparameters when available; otherwise we tune learning rate $\in \{10^{-4}, 5{\times}10^{-4}, 10^{-3}\}$, batch size $\in \{32, 64\}$, and hidden dimension $\in \{32, 64, 128\}$ on the validation set. All deep baselines train with Adam for up to 100 epochs (early stopping, patience $= 5$), matching AutoBricks' inner loop. Our reruns are averaged over three seeds; where published results use matching evaluation protocols, we report the better value.

Code-generation prompts. Codex and Claude Code receive forecasting instructions and task descriptions requesting a runnable PyTorch module mapping (B, T, N, C) inputs to (B, H, N, C) predictions.

Search space and comparison scope. In the main comparisons, AutoBricks uses the 38-brick registry, while the adapted NAS baselines use topology-specific nine-brick subsets. The restriction follows the memory limitations reported for our baseline implementations and configurations, and should not be interpreted as a universal limit of these methods. The subsets are listed below.

- **Grid topology** {CNN, GateCNN, ResConv, ResPlusConv, SpatialResNet, ConvLSTM, AttentionConvLSTM, LSTM, Transformer}.
- **Graph topology** {GCN, GAT, ChebGCN, DiffusionGCN, AdaptiveGCN, GRU, DilatedTCN, Transformer, TemporalAttention}.

The baseline adaptations retain method-specific search structures while using the listed operators. AutoBricks instantiates sampled candidates, allowing it to use the full registry in the evaluated setup. Differences in operator availability can affect forecasting accuracy, so the main comparison alone does not isolate the search mechanism. Matching the operator pool controls operator availability, while topology constraints, hyperparameter ranges, and search budgets must also be considered.

Operator-pool and random-assembly controls. We conduct two complementary comparisons on TaxiBJ and PEMS-BAY using GPT-5-mini for the AutoBricks variants. AutoBricks-9 uses the same topology-specific nine-operator subsets as the NAS baselines. Random-38 uses the same 38-brick registry, action constraints, search budget, and evaluation pipeline as AutoBricks-38, with random candidate assembly replacing the guided proposal mechanism.

Table 8: MAE of operator-pool and random-assembly controls. NAS-9 denotes AutoST on TaxiBJ and AutoSTF on PEMS-BAY.

Method	TaxiBJ	PEMS-BAY
NAS-9	16.78	1.55
AutoBricks-9	14.65	1.38
AutoBricks-38	**13.70**	**1.38**
Random-38	16.35	2.18

As shown in **Table 8**, AutoBricks-9 outperforms the NAS baseline on both tasks, supporting gains beyond access to additional operators. Expanding the registry improves TaxiBJ MAE, while PEMS-

BAY results remain equal at the reported precision. AutoBricks-38 also outperforms Random-38 under the matched assembly protocol, supporting the effectiveness of the guided search workflow beyond random modular assembly. These controls evaluate the combined workflow and do not separately attribute its gains to LLM reasoning, retrieval, or feedback.

Search budget and hardware. NAS baselines use the budget recommended in their original papers (typically 50–100 supernet epochs plus retraining). AutoBricks searches for at most 20 rounds per task, skips full training for failed candidates, and applies the stated early-stopping rule. All methods run on the same server (Intel Xeon Max 9462 @ 2.70 GHz, six NVIDIA A100 80GB GPUs); reported GPU hours include both search and final-model retraining for end-to-end comparability.

I LIMITATION DISCUSSION

AutoBricks currently searches architectures supported by a registry of 38 standardized bricks. Supporting additional operator families requires interface adaptation and compatibility checks, without updating LLM parameters. Search costs include LLM inference, sandbox execution, and candidate training and validation. Extending evaluation to ultra-large-scale urban deployments and streaming scenarios remains future work.