Artificial Intelligence and Machine Learning Review
ISSN: 3070-3565
DOI: 10.69987/AIMLR.2026.70210

LLM-Encoded Neural Architecture Descriptions for Cross-Model GPU Memory Prediction

Tyler Nguyen
Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA
tyler_n99@outlook.com

Keywords

GPU memory estimation; architecture descriptions; architecture-language embeddings; GPUMemNet; neural network training; cross-model prediction; resource management; out-of-memory classification.

Abstract

Accurate GPU memory prediction supports admission control, device selection, and capacity planning for deep learning workloads. Underestimation can cause out-of-memory failures after a job has already occupied an accelerator, whereas excessive safety margins reduce cluster utilization. This study evaluates an architecture-description approach on the GPUMemNet MLP, CNN, and Transformer training datasets. Each architecture is rendered as text from filename fields, scalar configuration variables, and its ordered layer inventory. A deterministic target-free architecture-language embedding is then fused with numeric and categorical design features. After removing 80 CNN rows with zero recorded target memory, the analysis contains 16,931 measured configurations. On a family-stratified 80/20 split, fused ExtraTrees achieved a mean absolute error of 1,059.9 MiB, a root mean square error of 2,335.0 MiB, an R2 of 0.9753, and 82.23% exact 4 GiB bucket accuracy. A numeric decision tree produced the lowest mixed-split MAE at 975.3 MiB, while fusion improved the ExtraTrees numeric baseline from 1,145.9 MiB to 1,059.9 MiB. Within-family fused ExtraTrees reached 12.8 MiB MAE for MLP, 858.3 MiB for CNN, and 914.8 MiB for Transformer. Leave-one-family-out performance was substantially weaker because the three families occupy different memory ranges and expose different allocation mechanisms. An auxiliary fused-feature OOM classifier reached 0.9702 F1. The findings show that architecture text is a useful semantic complement to numeric design variables, but dependable cross-family prediction still requires target-family observations and hardware-aware calibration.

Introduction

Deep learning training places pressure on GPU memory through parameters, gradients, optimizer state, activations, temporary workspaces, and framework-level allocations [1]. The exact balance changes with architecture and workload. Transformer attention and feed-forward blocks introduce sequence-dependent tensors [2], while modern language models also illustrate how model descriptions can carry meaningful structural information through tokenized representations [3]-[5]. For a scheduler, the practical question is not only whether a model is large, but whether a specific configuration will fit on an available device before execution begins.

Architecture-aware performance prediction has therefore become an important complement to analytical memory accounting. Language-based performance predictors have been used to initialize neural architecture search and to combine graph structure with textual model descriptions [6], [7]. Analytical estimators such as DNNMem model tensor lifetimes and runtime behavior directly [8], whereas a learned estimator can absorb relationships present in measured traces even when family-specific fields and logging conventions differ.

The prediction task is strongly nonlinear. Tree ensembles and boosted models are natural baselines for interacting configuration variables [9]-[12], and the underlying measurements are shaped by framework and optimizer choices such as PyTorch, TensorFlow, and Adam [13]-[15]. The three model families considered here span back-propagation-based multilayer perceptrons [16], convolutional designs that evolved from LeNet through AlexNet, ResNet, DenseNet, and EfficientNet [17]-[21], and attention-based Transformers [2]. Each family exposes different memory drivers: width and depth dominate many MLPs, activation geometry is central to CNNs, and sequence length, embedding width, head count, and feed-forward size are central to Transformers.

The need for reliable estimates grows with model scale [22], [23]. Memory-saving methods such as rematerialization, optimizer-state partitioning, and IO-aware attention reduce demand after a risky configuration has been identified [24]-[26], but schedulers still need a pre-execution estimate to decide whether such interventions are necessary. GPUMemNet provides measured MLP, CNN, and Transformer configurations for studying this problem and highlights both the promise and the transfer limits of learned GPU-memory estimators [27].

This study asks whether a compact architecture-language representation can improve memory prediction when it is combined with conventional design-time variables. The text representation is intentionally deterministic: architecture records are converted into structured descriptions, embedded without using memory labels, and fused with numeric and categorical features. This design isolates the contribution of architecture semantics while retaining the measured scalar quantities that directly determine allocation scale.

The contributions are fourfold. First, the study evaluates numeric, text-only, and fused representations on 16,931 measured configurations. Second, it compares mixed, within-family, and leave-one-family-out protocols rather than relying on a single favorable split. Third, it connects point prediction to device allocation through 4 GiB bucket metrics and to failure screening through OOM classification. Fourth, it analyzes transfer failure, memory-range error, and feature importance to identify where architecture-language features help and where additional coverage is required.

Literature Review

GPU Memory Estimation and Memory-Efficient Training

GPU memory estimation can be approached analytically, empirically, or through a hybrid of both. Analytical methods trace tensor creation and lifetime, producing interpretable estimates when graph structure and execution behavior are known [8]. Empirical estimators instead learn from completed runs and can absorb allocator behavior, framework overhead, and complex interactions that are difficult to express in closed form. GPUMemNet was developed around this empirical direction and provides model families, measurements, and analyses for training-aware resource management [27].

Memory-efficient training methods address a related but distinct problem. Checkmate selects rematerialization plans that trade additional computation for lower activation storage [24]. ZeRO partitions optimizer state, gradients, and parameters across devices [25], while FlashAttention reduces attention-related memory traffic through IO-aware tiling [26]. These techniques are corrective mechanisms: they reduce the footprint of a configuration after memory pressure is anticipated. A pre-execution predictor can therefore act as the front end that determines when a job should be admitted directly, moved to a larger device, or paired with a memory-saving strategy.

Architecture Descriptions and Predictive Representations

Text encoders such as BERT, large autoregressive language models, and unified text-to-text transformers established that token sequences can preserve semantic and structural regularities [3]-[5]. In architecture prediction, Jawahar et al. showed that language-model performance predictors can provide useful initializers for architecture search [6], and Selvam et al. examined how language information can complement structural representations for deep learning performance prediction [7]. The common idea is that layer order, repeated modules, normalization choices, activation names, and geometry fields form a structured language rather than a flat list of unrelated columns.

The present representation follows that idea without depending on a remote embedding service. Token and bigram TF-IDF features capture architecture vocabulary and local order, and truncated singular value decomposition produces a compact dense vector. The representation is target-free, so it does not encode measured memory values during fitting. Numeric features remain essential because parameter totals, activation counts, batch size, image dimensions, and sequence dimensions determine physical scale. The role of text is therefore complementary: it preserves family-specific semantics in a common description space.

Supervised Models and Evaluation Design

Linear regression provides a useful test of whether the transformed feature space is approximately additive, while single trees capture threshold-like behavior. Randomized forests reduce the instability of individual trees [10], gradient boosting builds nonlinear predictors through stage-wise residual fitting [11], and XGBoost provides an efficient implementation of boosted trees [9]. Scikit-learn supplies consistent preprocessing and evaluation components for these model classes [12]. Evaluating several inductive biases on the same feature matrices separates representation effects from model effects.

Evaluation protocol is equally important. A mixed split measures interpolation when all families are represented in training. A within-family split tests whether a model can learn a stable mapping for one architecture class. Leave-one-family-out evaluation measures transfer under a more severe shift in target range, feature support, and allocation mechanism. Reporting all three avoids conflating strong interpolation with general cross-model prediction.

GPU Capacity Forecasting, Transfer, and Operational Risk

Recent infrastructure studies place prediction inside broader resource-management workflows. Capacity forecasting with calibrated uncertainty has been used to bound provisioning risk in heterogeneous GPU clusters [28]. Cross-cloud transfer learning has examined workload and topology shift [29], while conformal demand envelopes have been used to control oversubscription risk [30]. Related work predicts salable GPU supply and instance readiness [31], forecasts multi-horizon demand with workload semantics [32], and addresses cold-start inference tenants with limited observations [33]. These studies reinforce a common operational principle: a point forecast is most useful when its domain, uncertainty, and decision cost are explicit.

Admission and inventory decisions extend the forecasting problem. Profit-aware spot admission control connects demand prediction to acceptance costs and service priorities [34], while power-aware inventory planning links job forecasts to infrastructure procurement and energy constraints [35]. Confidence-calibrated fusion offers a general strategy for combining signals with different reliability [36]. At the deployment level, hybrid cloud architectures trade local capacity against remote model execution [37], and AIOps systems use language models for incident triage and alert summarization [38]. Noisy-neighbor risk modeling further shows that resource interference can degrade observed performance even when nominal capacity appears sufficient [39].

Most of this literature operates at cluster, tenant, or incident scale. The present study addresses a complementary level: peak memory for an individual architecture configuration before training starts. It uses architecture language to bridge heterogeneous family metadata, evaluates transfer explicitly, and combines regression with OOM screening so that the output can support an admission decision rather than only a retrospective analysis.

Method

Dataset and Target Construction

The analysis uses the MLP, CNN, and Transformer portions of GPUMemNet [27]. The MLP file contains 3,000 configurations, the CNN file contains 9,000 configurations before filtering, and the Transformer file contains 5,011 configurations. The target is Max GPU Memory (MiB). Eighty CNN rows recorded a zero target and were removed because zero MiB cannot represent a completed training-memory measurement. The final regression set contains 16,931 rows: 3,000 MLP, 8,920 CNN, and 5,011 Transformer configurations.

OOM rows in CNN and Transformer were retained because they contain measured peak memory near the device limit. Their Status field was also used for the classification task. Runtime utilization counters - Avg GPUTL, Avg GRACT, Avg SMACT, Avg SMOCC, and Avg FP32A - were excluded because they are observed after execution begins and would not be available to a pre-admission estimator. Table 1 summarizes the filtered dataset and its target distribution.

Table 1. Dataset profile after target filtering.

Family	Rows used	Minimum (MiB)	Median (MiB)	Mean (MiB)	P90 (MiB)	Maximum (MiB)	OOM rows
CNN	8,920	1,451.0	18,419.0	21,238.6	39,863.0	40,369.0	2,797
MLP	3,000	1,443.0	1,576.0	1,743.3	2,241.2	4,925.0	0
Transformer	5,011	1,683.0	10,453.0	14,138.9	37,605.0	40,369.0	472

Architecture Description Encoder

Each row was converted into a structured architecture description from three sources: filename key-value fields, scalar design variables, and the ordered layer sequence in Activations-Params. The template records the architecture family, architecture label, activation function, depth, batch size, total parameters, total activations, the original filename fields, layer-type counts, and the first 80 layer tokens in order. This design preserves coarse scale, repeated modules, and local architectural sequence.

The architecture-language encoder uses token and bigram TF-IDF features with a vocabulary limit of 3,500 terms and a minimum document frequency of three. Truncated singular value decomposition projects the sparse text matrix to 48 dimensions. Neither Max GPU Memory nor Status participates in fitting the text encoder. The resulting representation is deterministic, compact, and independent of proprietary model versions.

Numeric, Categorical, and Fused Features

Numeric predictors include design-time scalar columns and numeric values parsed from filenames. Categorical predictors include activation function, architecture family, architecture label, and cleaned activation and architecture strings. Missing numeric values are median-imputed and standardized; missing categorical values receive a constant token and are one-hot encoded. The processed numeric/categorical matrix contains 87 columns. Concatenating it with the 48-dimensional text embedding produces 135 fused features. Table 2 lists the feature groups.

Table 2. Feature inventory used in the prediction pipeline.

Feature group	Count	Content
Numeric design	42	Depth, batch size, parameter and activation totals, estimated forward/backward memory, input and parameter sizes, embedding and feed-forward dimensions, parsed filename dimensions, and layer counts
Categorical design	5	Activation function, family, architecture label, cleaned activation string, and cleaned architecture string
Architecture text embedding	48	TF-IDF token and bigram representation projected with truncated SVD without memory labels
Target	1	Max GPU Memory (MiB)

Figure 1 shows the complete flow from raw architecture records to numeric, text, and fused model inputs. The same processed matrices were reused across regressors so that differences in performance reflect model behavior rather than inconsistent preprocessing.

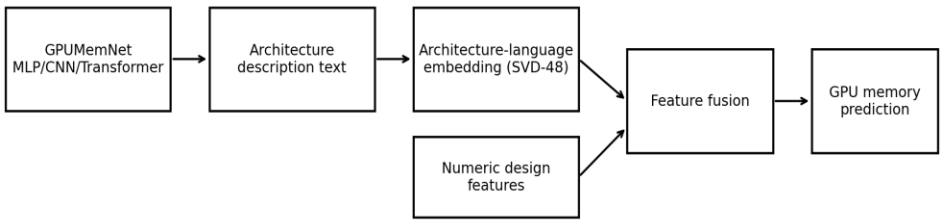

Figure 1. End-to-end architecture-description and numeric-feature fusion pipeline.

Models, Metrics, and Experimental Protocols

The regression target was transformed with log1p during training and mapped back to MiB with expm1. Negative predictions were clipped to positive values. Five regressors were compared: Ridge regression, elastic-net SGD regression, a single decision tree, ExtraTrees, and histogram-based XGBoost. The model set spans linear shrinkage, stochastic linear fitting, a single nonlinear rule system, randomized tree averaging, and boosted trees.

Performance was measured with MAE, RMSE, R2, MAPE, exact 4 GiB bucket accuracy, 4 GiB bucket MAE, under-prediction rate, and the 95th-percentile absolute error. The continuous metrics quantify numerical accuracy, while bucket accuracy reflects device-selection decisions in discrete memory bands. Under-prediction is reported because it is directly associated with scheduler risk.

Three regression protocols were used. The mixed protocol applies an 80/20 family-stratified split to all valid rows. The within-family protocol applies an 80/20 split separately to MLP, CNN, and Transformer. The leave-one-family-out protocol trains on two families and tests on the third. An auxiliary OOM classifier uses CNN and Transformer rows with Status labels and compares logistic regression, a decision tree, and ExtraTrees. All splits and models use random seed 42. Tables 3 and 4 summarize the protocols and fixed model settings.

Table 3. Experimental protocols.

Protocol	Rows	Split	Purpose
Mixed split	All valid rows	80/20, stratified by family	Regression comparison under interpolation
Within-family	One family at a time	80/20 random split	Family-specific estimation
Leave-one-family-out	Train on two families; test on the third	Deterministic family holdout	Cross-family transfer
OOM classification	CNN and Transformer rows with Status	80/20, stratified by OOM label	Pre-execution failure-risk detection
Target transform	All regression tasks	log1p for training; expm1 for prediction	Stable positive memory prediction

Table 4. Supervised model configurations.

Model	Configuration	Role
Ridge	alpha = 2.0	Linear shrinkage baseline
SGDRegressor	elastic-net; alpha = 1e-4; max_iter = 1,500	Fast stochastic linear baseline

Model	Configuration	Role
DecisionTree	max_depth = 16; min_samples_leaf = 5	Single nonlinear split model
ExtraTrees	10 trees; max_depth = 12; min_samples_leaf = 4	Randomized nonlinear ensemble
XGBoostHist	25 trees; depth = 4; learning_rate = 0.10	Compact histogram boosting baseline

Results and Discussion

Target Distribution and Mixed-Split Performance

The three architecture families occupy markedly different memory ranges. MLP is concentrated near 1.5-2.0 GiB, CNN spans almost the full device range, and Transformer has a broad middle-to-high distribution. Figure 2 visualizes this separation. The shift is central to interpretation: a mixed split is primarily an interpolation problem because every family appears in both training and test data, whereas leave-one-family-out evaluation requires extrapolation to a different target scale and feature regime.

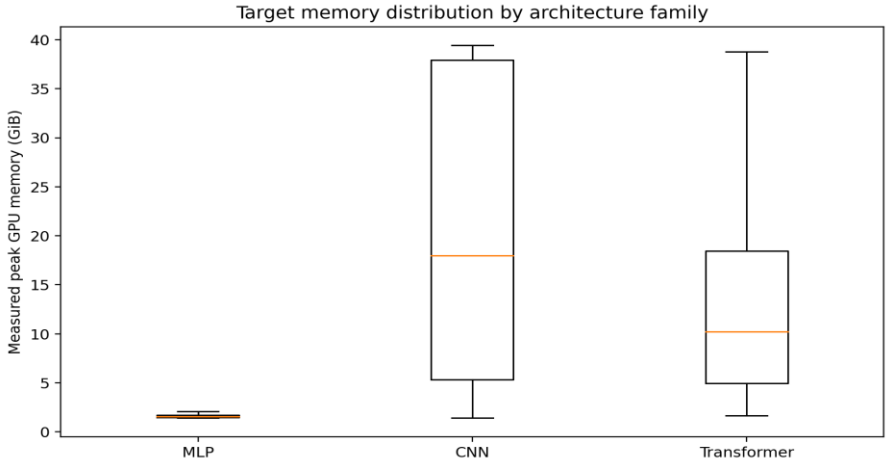

Figure 2. Distribution of measured peak GPU memory by architecture family.

Table 5 reports the complete mixed-split comparison. Numeric design features are already strong when paired with nonlinear trees. The numeric decision tree obtains the lowest MAE at 975.3 MiB and the highest R2 at 0.9812. The numeric ExtraTrees baseline reaches 1,145.9 MiB MAE. Adding the architecture-language embedding reduces ExtraTrees MAE to 1,059.9 MiB, lowers RMSE from 2,460.8 to 2,335.0 MiB, and increases exact 4 GiB bucket accuracy from 80.16% to 82.23%.

Table 5. Mixed 80/20 split regression results.

Feature set	Model	MAE (MiB)	RMSE (MiB)	R2	MAPE (%)	4 GiB bucket accuracy	Feature dimension
Numeric	Ridge	5,872.0	9,988.2	0.5489	38.6	0.4402	87
Numeric	SGDRegressor	5,904.2	10,193.2	0.5302	38.8	0.4461	87
Numeric	DecisionTree	975.3	2,037.8	0.9812	6.4066	0.8184	87
Numeric	ExtraTrees	1,145.9	2,460.8	0.9726	7.2237	0.8016	87
Numeric	XGBoostHist	8,297.4	11,878.8	0.3620	45.3	0.3652	87

Feature set	Model	MAE (MiB)	RMSE (MiB)	R2	MAPE (%)	4 GiB bucket accuracy	Feature dimension
Text	Ridge	8,899.0	12,212.1	0.3257	85.4	0.2861	48
Text	SGDRegressor	9,218.1	12,630.8	0.2787	89.2	0.2843	48
Text	DecisionTree	7,435.8	10,966.4	0.4563	74.0	0.3313	48
Text	ExtraTrees	7,824.3	10,773.7	0.4752	72.0	0.3047	48
Text	XGBoostHist	10,818.6	15,852.4	-0.1362	59.3	0.3348	48
Fused	Ridge	5,712.2	9,932.8	0.5539	36.3	0.4526	135
Fused	SGDRegressor	5,955.7	10,146.4	0.5345	38.1	0.4423	135
Fused	DecisionTree	1,038.5	2,127.5	0.9795	6.8940	0.8143	135
Fused	ExtraTrees	1,059.9	2,335.0	0.9753	6.5900	0.8223	135
Fused	XGBoostHist	8,288.4	11,830.6	0.3672	45.4	0.3634	135

The linear models remain several GiB away from the target on average, indicating that additive relationships in the transformed feature space are insufficient. The compact XGBoost configuration also performs poorly, which suggests that its shallow boosted ensemble does not capture the relevant interactions under the fixed settings. Text-only models are weaker than numeric models because tokenized architecture descriptions do not replace physical scale variables such as activation count, image geometry, and sequence dimensions.

Figure 3 places the feature sets side by side for each regressor. Fusion is most useful for ExtraTrees and modestly improves Ridge, but it does not improve every model. This pattern is consistent with a complementary representation: architecture text contributes family labels, ordered layer cues, and repeated-module semantics, while numeric features carry the main memory scale.

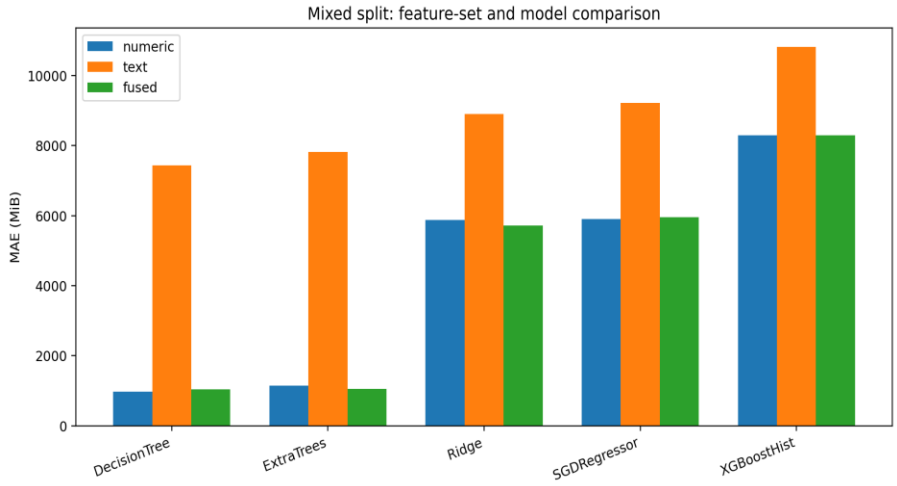

Figure 3. Mixed-split comparison of feature sets and supervised regressors.

Figure 4 compares fused ExtraTrees predictions with measured memory. The points form a dense diagonal band at low and middle memory levels, with larger downward deviations among high-memory CNN and Transformer cases. Many of these configurations lie near the device ceiling, where OOM behavior and allocator limits compress the

recorded peak. Even in this region, the 82.23% exact bucket accuracy indicates that the estimator often preserves the allocation class when the MiB estimate is imperfect.

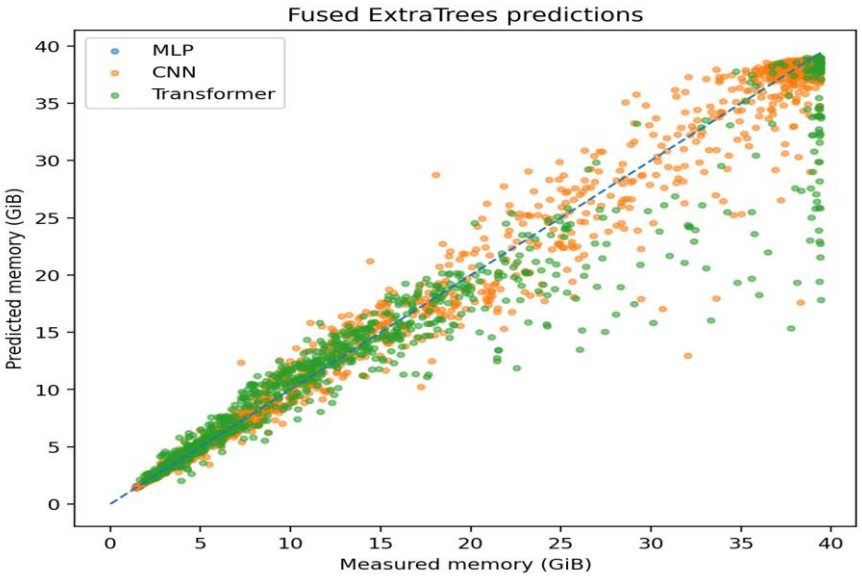

Figure 4. Fused ExtraTrees predictions versus measured memory on the mixed test split.

Within-Family Estimation

Table 6 shows that family-specific models are substantially more accurate than cross-family models. MLP memory is highly regular: numeric ExtraTrees reaches 12.0 MiB MAE and fused ExtraTrees reaches 12.8 MiB. CNN fused ExtraTrees improves on its numeric counterpart, reducing MAE from 871.5 to 858.3 MiB and increasing 4 GiB bucket accuracy from 84.47% to 84.98%. Transformer numeric and fused results are close, with 899.2 and 914.8 MiB MAE, respectively; fusion produces a slightly higher R2 but a lower bucket accuracy.

Table 6. Within-family ExtraTrees regression results.

Family	Feature set	MAE (MiB)	RMSE (MiB)	R2	MAPE (%)	4 GiB bucket accuracy	P95 absolute error (MiB)
MLP	Numeric	12.0	26.5	0.9959	0.6093	0.9967	40.3
MLP	Text	241.6	401.1	0.0616	12.5	0.9950	790.6
MLP	Fused	12.8	28.0	0.9954	0.6489	0.9983	47.7
CNN	Numeric	871.5	1,527.3	0.9903	4.9801	0.8447	3,073.2
CNN	Text	9,481.7	11,600.4	0.4385	84.2	0.1469	21,240.2
CNN	Fused	858.3	1,493.3	0.9907	5.0601	0.8498	2,968.3
Transformer	Numeric	899.2	1,683.1	0.9787	6.6774	0.8106	3,207.8
Transformer	Text	8,114.7	11,104.1	0.0735	81.6	0.1725	26,054.0
Transformer	Fused	914.8	1,669.2	0.9791	6.7689	0.8046	3,498.2

The results support family-aware deployment. Once a scheduler has collected enough observations for a target class, tree models can learn the corresponding memory mechanisms with high precision. Text embeddings are most beneficial when the numeric representation leaves useful architecture semantics unresolved; they add little when scalar design variables already describe the dominant memory path.

Cross-Family Transfer

Leave-one-family-out transfer is much harder. Table 7 reports negative R2 for every held-out family and feature set. When MLP is held out, numeric and fused models learn a high-memory prior from CNN and Transformer and overpredict the low-memory MLP range. Text-only transfer lowers MLP MAE from 16,591.5 to 6,803.5 MiB because family and layer vocabulary identify simpler architectures, but the error remains too large for dependable placement.

Table 7. Leave-one-family-out cross-model transfer results.

Held-out family	Feature set	MAE (MiB)	RMSE (MiB)	R2	MAPE (%)	4 GiB bucket accuracy	Under-prediction rate
MLP	Numeric	16,591.5	16,713.3	-1264.1	1010.1	0.0000	0.0000
MLP	Text	6,803.5	7,730.1	-269.6	411.2	0.0550	0.0010
MLP	Fused	13,566.0	13,752.9	-855.6	822.3	0.0000	0.0000
CNN	Numeric	17,604.8	22,912.9	-1.2207	69.6	0.1816	0.8219
CNN	Text	18,903.7	24,308.0	-1.4993	72.7	0.1905	0.9175
CNN	Fused	18,156.0	23,377.9	-1.3117	69.0	0.1948	0.9166
Transformer	Numeric	9,579.1	14,793.3	-0.6438	49.0	0.3079	0.7597
Transformer	Text	9,626.8	14,329.2	-0.5422	58.9	0.2321	0.7212
Transformer	Fused	9,623.8	14,838.4	-0.6538	48.6	0.3233	0.7743

Holding out CNN removes the feature-map regime from training, producing high error and under-prediction rates above 82%. Holding out Transformer is less severe in bucket terms; fused features reach 32.33% exact bucket accuracy compared with 30.79% for numeric features. Figure 5 summarizes the MAE gap. The broader lesson matches transfer-oriented capacity research [29], [33]: semantic representations can improve alignment, but they cannot create target-scale calibration when the training data contain no comparable family.

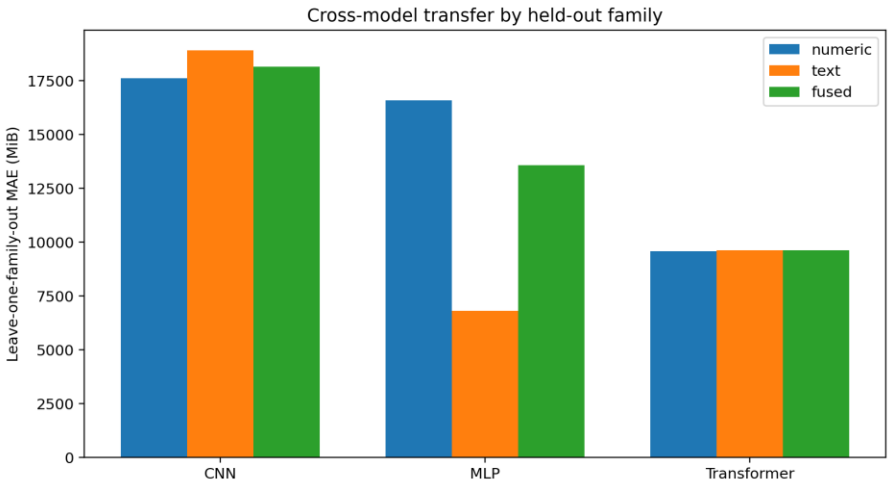

Figure 5. Leave-one-family-out MAE by held-out family and feature set.

OOM Classification and Error Concentration

The OOM task is easier than exact memory regression because failed runs are concentrated near the upper memory limit. Table 8 shows that fused logistic regression reaches 98.56% accuracy, 0.9954 recall, 0.9702 F1, and 0.9995 ROC-AUC. The decision tree performs similarly. ExtraTrees has lower precision but retains 0.9771 recall, making it suitable for conservative screening when missed OOM cases are more costly than false alarms.

Table 8. OOM classification on CNN and Transformer.

Model	Accuracy	Precision	Recall	F1	ROC-AUC	Training rows	Test rows
LogisticRegression	0.9856	0.9462	0.9954	0.9702	0.9995	11,144	2,787
DecisionTree	0.9846	0.9526	0.9832	0.9676	0.9884	11,144	2,787
ExtraTrees	0.9670	0.8925	0.9771	0.9328	0.9963	11,144	2,787

Table 9 and Figure 6 provide the ExtraTrees confusion matrix. Of 654 OOM cases in the test set, 639 were detected and 15 were missed. Seventy-seven non-OOM runs were flagged conservatively. In an admission controller, the threshold can be moved toward recall when the cost of a crash exceeds the cost of assigning a larger device. This risk-based use is consistent with uncertainty-bounded oversubscription and admission-control work [28], [30], [34].

Table 9. ExtraTrees OOM confusion matrix.

Actual class	Predicted non-OOM	Predicted OOM
Actual non-OOM	2,056	77
Actual OOM	15	639

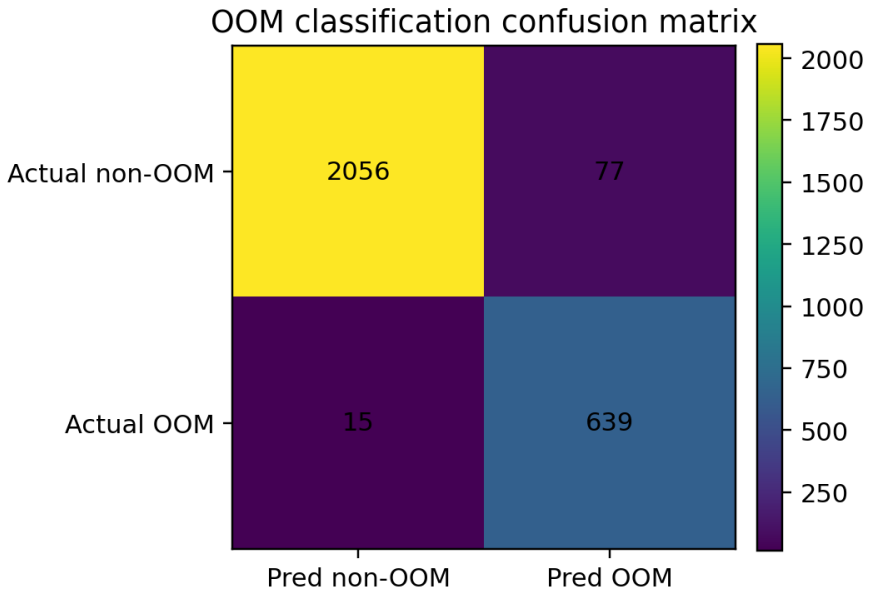

Figure 6. OOM confusion matrix for the fused ExtraTrees classifier.

Table 10 locates the main regression errors. MLP remains below 50 MiB MAE in every observed range. CNN error rises from 34.8 MiB below 2 GiB to 2,426.7 MiB in the 16-32 GiB band, then falls above 32 GiB because many high-memory CNN runs cluster near the same device ceiling. Transformer error is moderate below 8 GiB but reaches 5,549.7 MiB above 32 GiB. Figure 7 makes this concentration visible.

Table 10. Fused ExtraTrees mixed-test error by family and memory range.

Family	Memory range	Rows	MAE (MiB)	RMSE (MiB)	Median absolute error (MiB)
CNN	0-2 GiB	90	34.8	55.7	16.6
CNN	2-4 GiB	250	180.3	243.7	137.4

Family	Memory range	Rows	MAE (MiB)	RMSE (MiB)	Median absolute error (MiB)
CNN	4-8 GiB	237	440.3	646.3	351.6
CNN	8-16 GiB	241	1,191.4	1,551.3	940.0
CNN	16-32 GiB	248	2,426.7	3,137.8	2,048.4
CNN	>32 GiB	718	1,037.4	1,995.0	636.9
MLP	0-2 GiB	526	9.0516	15.8	5.8644
MLP	2-4 GiB	72	47.0	58.6	36.8
MLP	4-8 GiB	2	48.6	54.5	48.6
Transformer	0-2 GiB	8	262.5	370.3	101.4
Transformer	2-4 GiB	194	323.0	463.8	227.1
Transformer	4-8 GiB	209	726.3	979.0	506.4
Transformer	8-16 GiB	290	1,167.0	1,432.3	1,001.7
Transformer	16-32 GiB	174	3,045.1	4,343.5	1,857.4
Transformer	>32 GiB	128	5,549.7	8,078.1	2,166.2

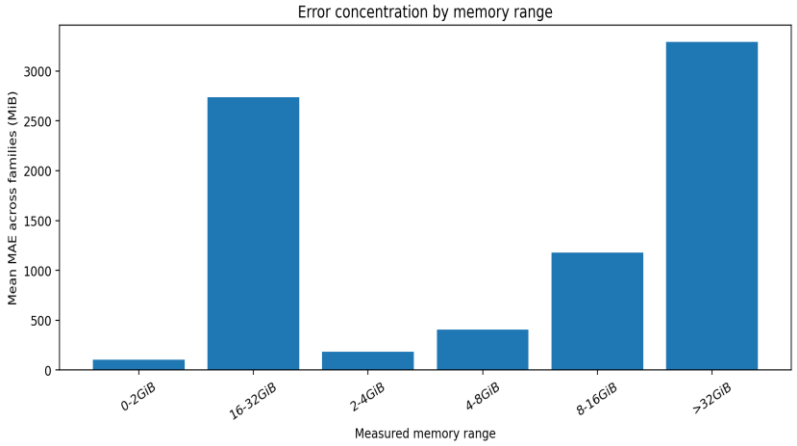

Figure 7. Mean absolute error by measured memory range.

The high-memory bands require a safety mechanism beyond a single point estimate. A practical scheduler can add a family- and range-dependent margin, use the OOM probability as a guardrail, or fit prediction intervals. Calibrated fusion methods provide a general framework for combining the regressor, classifier, and uncertainty signals [36]. For Transformer jobs near the ceiling, this approach is preferable to treating a point prediction as an exact capacity requirement.

Feature Importance and Operational Implications

Table 11 lists the highest ExtraTrees importances from the numeric and one-hot component. A categorical indicator is dominant, reflecting the strong separation among families and architecture labels. Forward/backward pass size, estimated total size, parameter size, embedding size, input size, activation count, accumulated parameters, feed-forward hidden size, total parameters, and batch size also rank highly. These variables align with the physical sources of memory: parameters and optimizer state scale with trainable weights, activations scale with batch and geometry, and Transformer allocations scale with embedding and feed-forward dimensions.

Table 11. Top numeric and one-hot features by ExtraTrees importance.

Feature	Importance
onehot_categorical_15	0.4132
Forward/Backward Pass Size (MB)	0.1527
Estimated Total Size (MB)	0.1044
Params Size (MB)	0.0543
Embedding Size	0.0386
Input Size (MB)	0.0374
Total Activations	0.0365
Accumulated Params	0.0308
fn__ff_hidden_size	0.0246
fn__embedding_size	0.0225
Total Parameters	0.0114
Batch Size	0.0098

The operational value of the fused model is not that it replaces analytical understanding, but that it turns heterogeneous configuration metadata into a stable screening signal. The regression output estimates expected peak demand, the bucket output supports device selection, and the classifier identifies configurations with elevated failure risk. Capacity and supply forecasts can provide the cluster-level envelope [28], [31], [32], [35], while per-configuration memory prediction determines which jobs fit inside that envelope.

Deployment should also account for the surrounding system. Hybrid cloud routing changes the available device pool and the cost of offloading [37]. Incident-triage systems can connect prediction failures with logs and operational evidence [38], and noisy-neighbor effects can invalidate a capacity estimate when co-located workloads interfere with execution [39]. These considerations favor a monitored workflow in which memory estimates are recalibrated by hardware type, framework version, precision mode, and observed drift.

Limitations

The architecture-language encoder is compact and deterministic rather than a large pretrained code model. It captures vocabulary, local token order, and low-rank document structure, but it does not model full source-code semantics. A larger encoder could represent control flow, tensor reuse, and custom operators more richly, although it would introduce model-download, version, and inference-cost dependencies.

Cross-family extrapolation remains the main statistical limitation. The leave-one-family-out results show that semantic similarity does not resolve severe target-range shift. Accurate deployment therefore requires at least a small calibration set from the target family, particularly for CNN feature-map behavior and high-memory Transformer configurations.

OOM targets are censored. The recorded peak indicates memory reached before failure, not the unknown amount required to complete training. Regression treats this observed peak as the target and classification models the failure event separately. A censored-regression or survival-style formulation could estimate the latent requirement beyond the device ceiling more directly.

The measurements also reflect the hardware, framework, precision, optimizer, and allocator conditions under which GPUMemNet was collected. Estimates should be recalibrated before use on different GPU generations, mixed-precision settings, activation-checkpointing policies, optimizer-state layouts, or framework versions. The descriptions do not include dataloader behavior, custom kernels, allocator fragmentation, or all temporary workspaces, each of which can affect observed peak memory.

Conclusion

This study evaluated architecture-language descriptions for GPU memory prediction across MLP, CNN, and Transformer training configurations. Tree-based models over design-time variables were the strongest overall predictors. On the mixed split, fused ExtraTrees achieved 1,059.9 MiB MAE, 0.9753 R2, and 82.23% exact 4 GiB bucket accuracy, improving the ExtraTrees numeric baseline. Family-specific models were highly accurate, including 12.8 MiB MAE for MLP, 858.3 MiB for CNN, and 914.8 MiB for Transformer with fused features.

Architecture text did not replace numeric memory variables, but it contributed complementary semantics and improved selected ensemble and transfer results. The weak leave-one-family-out performance establishes an important boundary: architecture descriptions cannot supply missing target-scale evidence. The fused OOM classifier adds a practical second output, reaching 0.9702 F1 and allowing schedulers to distinguish expected demand from failure risk.

A robust deployment should combine family-aware regression, an OOM guardrail, explicit safety margins, and periodic calibration for hardware and workload drift. Within that workflow, architecture-language embeddings provide a reproducible way to unify heterogeneous model metadata and improve pre-execution resource decisions.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[2] A. Vaswani et al., “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998-6008.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019, pp. 4171-4186.

[4] T. B. Brown et al., “Language models are few-shot learners,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 1877-1901.

[5] C. Raffel et al., “Exploring the limits of transfer learning with a unified text-to-text transformer,” J. Mach. Learn. Res., vol. 21, no. 140, pp. 1-67, 2020.

[6] G. Jawahar, M. Abdul-Mageed, and L. Lakshmanan, “LLM performance predictors are good initializers for architecture search,” in Findings of the Association for Computational Linguistics: ACL, 2024, pp. 10630-10648.

[7] K. P. Selvam, E. M. Brorsson, and J. X. Zheng, “Can LLMs enhance performance prediction for deep learning models?” in Proc. ICML Workshop on Automated Machine Learning for Neural Architecture Search, 2024.

[8] C. Wang, Z. Wen, R. Zhang, P. Xu, and Y. Jiang, "GPU Memory Requirement Prediction for Deep Learning Task Based on Bidirectional Gated Recurrent Unit Optimization Transformer," in 2025 5th International Conference on Artificial Intelligence, Virtual Reality and Visualization (AIVRV), Chengdu, China, 2025, doi: 10.1109/AIVRV67401.2025.11350369.

[9] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785-794.

[10] L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[11] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” Annals of Statistics, vol. 29, no. 5, pp. 1189-1232, 2001.

[12] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825-2830, 2011.

[13] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8024-8035.

[14] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in Proc. USENIX Symp. Operating Systems Design and Implementation, 2016, pp. 265-283.

[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learning Representations, 2015.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” Nature, vol. 323, pp. 533-536, 1986.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097-1105.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2016, pp. 770-778.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2017, pp. 4700-4708.

[21] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. Int. Conf. Machine Learning, 2019, pp. 6105-6114.

[22] J. Kaplan et al., “Scaling laws for neural language models,” arXiv:2001.08361, 2020.

[23] J. Hoffmann et al., “Training compute-optimal large language models,” arXiv:2203.15556, 2022.

[24] P. Jain, A. Jain, T. Luo, and S. Abbeel, “Checkmate: Breaking the memory wall with optimal tensor rematerialization,” in Proc. Machine Learning and Systems, 2020, pp. 497-511.

[25] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, “ZeRO: Memory optimizations toward training trillion parameter models,” in Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1-16.

[26] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Re, “FlashAttention: Fast and memory-efficient exact attention with IO-awareness,” in Proc. Adv. Neural Inf. Process. Syst., 2022, pp. 16344-16359.

[27] E. Yousefzadeh-Asl-Miandoab, R. Karimzadeh, D. Yorulmaz, B. Ibragimov, and P. Tozun, “GPU memory and utilization estimation for training-aware resource management: Opportunities and limitations,” in Proc. Sixth European Workshop on Machine Learning and Systems (EuroMLSys ’26), 2026, pp. 127-138, doi: 10.1145/3805621.3807621.

[28] S. He, H. Tu, and I. Liu, “Safe PD capacity forecasting with time-series foundation models and calibrated uncertainty for heterogeneous GPU clusters,” J. Adv. Comput. Syst., vol. 3, no. 4, pp. 48-66, Apr. 2023, doi: 10.69987/JACS.2023.30404.

[29] S. He, X. Chang, and E. Sun, “Cross-cloud transfer learning for AI training capacity forecasting under workload and topology distribution shift,” J. Adv. Comput. Syst., vol. 4, no. 1, pp. 100-120, Jan. 2024, doi: 10.69987/JACS.2024.40108.

[30] S. Chen, S. He, and E. Sun, “Risk-bounded GPU resource oversubscription via conformal demand envelopes in production AI clusters,” J. Adv. Comput. Syst., vol. 4, no. 5, pp. 119-134, May 2024, doi: 10.69987/JACS.2024.40509.

[31] H. Tu, S. Zhao, and S. He, “LLM-augmented salable GPU supply forecasting for disaggregated recommendation serving: Predicting instance readiness, scheduling delay, and capacity risk,” J. Adv. Comput. Syst., vol. 4, no. 9, pp. 85-102, Sep. 2024, doi: 10.69987/JACS.2024.40908.

[32] S. Zhao, J. Bai, and D. Roberson, “Multi-horizon GPU demand forecasting with workload semantics and operational risk curves: An empirical study on Alibaba Clusterdata GPU trace,” J. Technol. Informatics Eng., vol. 4, no. 3, pp. 544-571, Dec. 2025, doi: 10.51903/jtie.v4i3.498.

[33] S. He and A. Qian, “Few-shot cold-start workload forecasting for new AI inference tenants with time-series foundation models,” J. Technol. Informatics Eng., vol. 4, no. 1, pp. 306-324, Apr. 2025, doi: 10.51903/jtie.v4i1.546.

[34] S. Zhao, Y. Ren, and X. Chang, “Profit-aware spot GPU admission control with cost-sensitive loss and evidence-grounded policy memos for AI workload supply-demand matching,” J. Technol. Informatics Eng., vol. 5, no. 2, pp. 45-59, Jun. 2026, doi: 10.51903/jtie.v5i2.545.

[35] S. He, “Power-aware inventory planning for AI infrastructure using job-level forecasting and LLM workload explanations,” J. Technol. Informatics Eng., vol. 5, no. 1, pp. 341-359, Apr. 2026, doi: 10.51903/jtie.v5i1.548.

[36] Q. Xin, “Uncertainty-aware late fusion for 3D perception (confidence calibration + fusion rule learning),” J. Technol. Informatics Eng., vol. 4, no. 1, pp. 215-238, Apr. 2025, doi: 10.51903/jtie.v4i1.485.

[37] Q. Xin, “Hybrid cloud architecture for efficient and cost-effective large language model deployment,” J. Inf. Syst. Informatics, vol. 7, no. 3, pp. 2182-2195, Sep. 2025, doi: 10.51519/journalisi.v7i3.1170.

[38] G. Liu, C. Li, and E. Zhang, “OpsLLM for cloud incident triage: Bilingual RAG-based root cause analysis and alert summarization for AI infrastructure operations,” J. Adv. Comput. Syst., vol. 4, no. 4, pp. 97-111, Apr. 2024, doi: 10.69987/JACS.2024.40408.

[39] J. Nie and D. Zheng, “Noisy-neighbor-aware VM degradation risk modeling with unsupervised residual fusion,” J. Adv. Comput. Syst., vol. 4, no. 4, pp. 112-123, Apr. 2024, doi: 10.69987/JACS.2024.40409.