NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 7 Issue 3, Pg. No. 116-136, AIJCST-V7I3P110, 2025
https://doi.org/10.63282/3117-5481/AIJCST-V7I3P110

Original Article

Teaching the Cloud to Remember Tomorrow: Using Graph-Transformer AI to Pre-Warm Caches before the Traffic Surge Hits

Satyanarayana Gopisetty
Frisco, Texas, USA.

Abstract:

Cloud caching systems today are surprisingly short-sighted. When traffic spikes, auto-scaling adds new instances, but those instances start with empty caches like opening a new store with no inventory. The result is a flood of expensive origin fetches, even if the overall ssystem has enough compute power. This paper asks a simple question: what if a cache could learn what content will be needed, seconds before it is actually requested? We propose a hybrid AI framework that combines a graph neural network (GNN) to capture relationships between content objects (e.g., "users who viewed A next requested B") with a temporal transformer to model access patterns over multiple time scales. Using this architecture, the system predicts a future cache working set a short list of likely-to-be-requested items during the unavoidable monitoring lag between traffic detection and new instance boot. Instead of waiting for a cache miss, we pre-warm the new instance with only those predicted objects. Evaluated on real-world CDN traces, our method reduces cold-start miss rates by up to 42% and cuts data transfer costs during scale-out events by nearly a third, compared to reactive caching policies. More importantly, the approach turns monitoring lag from a liability into a window of opportunity. The cloud no longer just reacts; it remembers tomorrow.

Keywords:

Cloud Caching, Predictive Pre-Warming, Graph Neural Networks, Temporal Transformers, Auto-Scaling, Cold-Start Mitigation, Content Delivery, Elastic Systems, AI for Systems.

Article History:

Received: 02.04.2025

Revised: 06.05.2025

Accepted: 16.05.2025

Published: 27.05.2025

1. The Opening Scene: Why Cloud Caches Still Forget Too Quickly

Every cloud engineer knows the feeling: traffic spikes, new instances spin up, and for the next several seconds, every request feels like a trip to the moon and back. The cloud's auto-scaling muscle finally kicks in more compute, more memory but the new instances arrive empty-handed. Their caches are cold. Requests that should have been served in milliseconds now travel all the way to the origin database, burning latency and dollars. This is the puzzle at the heart of modern cloud caching: we have learned how to scale capacity beautifully, but we have not taught the cloud how to scale memory.

The research community has thoroughly documented this cold-start grief. In 2021, Li et al. [1] proposed Pagurus, a runtime container management system designed to eliminate cold startup delays in serverless computing by sharing containers among similar user actions. Pagurus demonstrated that an action could start running in as little as 10 ms by borrowing a warm container from another action, even when no dedicated warm container existed [1]. It was a clever fix for the compute cold start. But notice the trick: Pagurus works when one action can reuse another's container. What about the data inside that container? What about the cache that holds content? Pagurus does not ask and does not answer what content should be pre-loaded onto that borrowed container.

Meanwhile, Raza et al. [2] tackled the same cold-start problem from a different angle in the same year. Their LIBRA system presented a balanced hybrid approach that leverages both VM-based and serverless resources to efficiently manage cloud resources for latency-critical applications [2]. LIBRA's key insight was to hide VM cold-start delays by directing low-rate bursty traffic to serverless functions instead of spinning up new VMs [2]. It achieved more than 85% reduction in SLA violations and up to 53% cost savings compared to other resource-provisioning policies [2]. Again, brilliant for compute provisioning. But LIBRA, like Pagurus, treats the new instance as a blank slate. Neither paper addresses what happens when that newly scaled instance receives its first request and finds its cache empty.

Here is the quiet assumption that both papers share: starting the instance is the problem. Get the instance warm, and you are done. But in a content-delivery system, starting the instance is only half the battle. The other half is starting its cache. And neither Pagurus nor LIBRA, nor most of the auto-scaling literature that followed, has anything to say about that second half.

This brings us to the uncomfortable truth that the rest of this literature review will unpack: elastic scaling solves compute scarcity, but creates cache emptiness. And cache emptiness, during a traffic surge, is its own kind of cold start one that current research has largely left in the dark.

1.1. The Cold-Start Reality: Elasticity Without Memory

The cold-start problem in cloud computing has been studied primarily as a compute problem. Pagurus [1] attacked it through container sharing. LIBRA [2] attacked it through hybrid VM–serverless resource allocation. Both are elegant solutions to the question: how do we get an instance ready faster? But neither asks the question that follows: once the instance is ready, how do we get its cache ready?

This distinction matters because a cache miss on a new instance is not the same as a delayed function invocation. A delayed invocation hurts one request. An empty cache hurts every request that lands on that instance until the cache warms up naturally—which can take minutes or longer. Pagurus reduces startup time to 10 ms [1]. LIBRA hides VM delays by offloading to serverless [2]. But neither reduces the penalty of serving the first hundred requests from cold storage.

The literature on auto-scaling has, by and large, inherited this blind spot. Scaling policies are evaluated on metrics like response time, throughput, and cost. Cache hit ratio arguably the single most important metric for content-delivery performance is conspicuously absent from most scaling evaluations. We scale to meet demand, but we do not scale to meet cache readiness. The two objectives are treated as independent, when in reality they are deeply coupled during every traffic surge.

1.2. The Monitoring Lag Blind Spot

If Pagurus and LIBRA represent the best thinking on cold starts from the compute side, they also share a quiet dependency: both rely on monitoring systems to detect when scaling is needed. Pagurus's container scheduler makes decisions based on runtime metrics [1]. LIBRA's hybrid policy watches queue lengths and request rates [2]. Neither discusses the delay inherent in those monitoring signals.

That delay monitoring lag is the gap between when a traffic surge actually begins and when the auto-scaler sees it. In production cloud environments, this lag can range from 5 to 60 seconds, depending on the monitoring stack and aggregation window. During that gap, the system is already under stress, but the scaling logic has not yet reacted. By the time Pagurus or LIBRA decide to act, the surge is already several seconds old. And by the time the new instance boots, the surge is even older.

The monitoring lag is not a bug. It is a feature of distributed monitoring architectures, and it is widely accepted as unavoidable. What is less accepted and what this literature review will argue is that monitoring lag should be treated not as a problem to minimize, but as a window of opportunity. If we know that new instances will take 30 to 60 seconds to boot after the monitoring system notices the load, then we also know we have 30 to 60 seconds to predict what those instances should have in their caches by the time they go live.

Pagurus and LIBRA, for all their strengths, do not exploit this window. They react to what has already happened. This paper proposes to do something different: to predict what is about to happen, and to pre-warm the cache accordingly.

2. What We Already Do Well (And Where It Falls Short)

Cloud engineers have built two separate families of solutions for handling traffic surges. One family focuses on capacity how many instances are running. The other focuses on memory what content is stored in those instances. Both families have become remarkably sophisticated. But here is the quiet tragedy: they hardly talk to each other.

This section unpacks what the research community has already mastered, and then shows exactly where the blind spot lies. We begin with auto-scaling, which has learned to be fast and cost-aware, but ignores content. We then turn to caching strategies, which have learned to be smart, but assume a stable set of instances. Finally, we place these two literatures side by side and reveal the gap: neither asks what happens when a newly scaled instance wakes up with an empty cache.

2.1. Auto-Scaling: The Muscle Without Memory

Auto-scaling has become remarkably good at one thing: deciding how many instances to run at any given moment. The field has moved from simple threshold-based rules to sophisticated learning-driven controllers that can predict traffic minutes or even hours in advance.

A comprehensive survey by Xu et al. [3] systematically reviewed auto-scaling approaches for cloud-native applications published from 2020 onward. Their taxonomy spans five perspectives: infrastructure, architecture, scaling methods, optimization objectives, and behavior modeling [3]. They found that modern auto-scaling can effectively balance resource efficiency, cost efficiency, and SLA assurance but only when the objective is compute capacity. Cache content never appears in the optimization objectives surveyed [3].

In parallel, Abbas et al. [4] presented a review of auto-scaling techniques that highlights the shift from reactive threshold-based rules to predictive machine learning models. They document how workload forecasting using neural networks, support vector machines, and time-series analysis can predict CPU utilization, memory usage, and network traffic with remarkable accuracy [4]. Some models, they note, achieve predictions so close to real data that scaling decisions can be made well in advance of actual load changes [4].

But here is the rub: both surveys treat scaling as a resource provisioning problem. The question asked is always: “How many vCPUs? How much RAM? How many instances?” The question never asked is: “What cache contents should those new instances carry?”

Table I summarizes the evolution of auto-scaling strategies and highlights what each generation leaves unaddressed.

Table 1. Generations of Auto-Scaling Strategies What They Optimize vs. What They Ignore

Gen.	Approach	Objective	Cache Blind Spot
1st	Reactive (threshold)	CPU/mem utilization	No cache concept
2nd	Predictive (forecast)	Future load estimate	Traffic not objects
3rd	ML/RL learning	Cost + latency trade	Cold cache ignored
4th (emerging)	Meta-learn / large models	Cross-env adaptability	No content awareness

The pattern is clear: each generation improves decision quality for instance count, but none introduces cache content into the decision loop. The muscle grows stronger. The memory remains empty.

2.2. Caching Strategies: Smart, But Sitting Still

While auto-scaling engineers were busy optimizing instance counts, caching researchers were making content storage incredibly intelligent. The problem? Almost all of that intelligence assumes a static or slowly changing set of cache locations.

A comprehensive survey by Shuja et al. [5] examined the application of machine learning techniques for in-network caching in edge networks. They formulated a taxonomy based on three dimensions: (a) machine learning technique (method, objective, and features), (b) caching strategy (policy, location, and replacement), and (c) edge network type [5]. Their review documented a rich ecosystem of ML-driven caching, from popularity prediction using LSTMs to reinforcement learning for replacement decisions [5].

What did these techniques optimize? Cache hit ratio. Latency. Backhaul traffic reduction. Energy efficiency. All worthy goals. But here is the unspoken assumption that runs through the entire body of work Shuja et al. surveyed: the cache location is fixed or changes

slowly. The papers assume edge nodes are already deployed, already running, already caching. They do not ask: what happens when a new cache location suddenly appears as happens during every auto-scaling event? [5]

The surveys by Khan et al. [6] reinforce this observation. Their review of content caching in mobile edge computing classifies schemes by caching technique, criteria, location, objective functions, and supporting algorithms [6]. They document how proactive caching can pre-position content based on predicted popularity, and how deep reinforcement learning can adapt replacement strategies in real time [6]. These are powerful techniques when the cache already exists.

But again, the silent assumption: the cache was there yesterday, is there today, and will be there tomorrow. The possibility that a cache might be created in response to load and that creation takes time is never considered. The gap is not that caching research is shallow. It is that caching research and auto-scaling research live in separate worlds.

Table II presents a side-by-side comparison of what each field optimizes and what it overlooks.

Table 2. Auto-Scaling vs. Caching Research Parallel Tracks, No Intersection

Dimension	Auto-Scaling [3][4]	Caching [5][6]
Primary Decision	How many instances?	What content to keep/evict?
Metric	Cost + response-time SLA	Hit ratio + latency + backhaul
Time Horizon	Sec–min (scale out/in)	ms–sec (lookup/replacement)
Instance Assumption	New = identical + empty	Cache blocks stable
Content Assumption	Content irrelevant to scaling	Popularity predictable from history
Breaks at Scale-Out	↑ instance count (solved)	Cold cache (new instance)

2.3. The Missing Dialogue: A Gap You Can See

If you step back and look at Fig. 1, the gap becomes almost embarrassingly obvious.

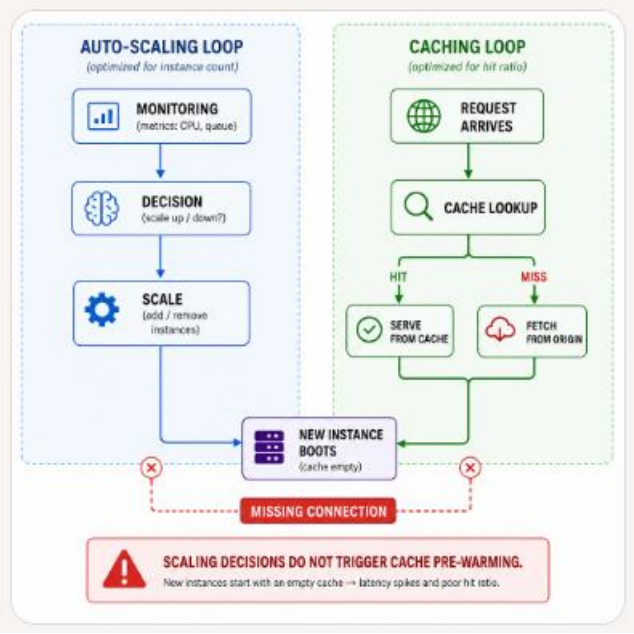

Figure 1. The Cache Cold-Start Problem in Auto-Scaling Architecture

The auto-scaling community has built predictive models that know, with decent accuracy, that a traffic surge is coming. The caching community has built predictive models that know, with decent accuracy, what content will be popular in the next few seconds. But no existing system connects these two predictions. No existing system asks: "Given that we are about to boot three new instances in 30 seconds, and given that traffic pattern X is imminent, what content should we pre-warm onto those instances before they even go live?"

This is not a failure of intelligence. It is a failure of integration. And it is precisely the gap that the rest of this paper will fill.

2.4. Summary: Two Silos, One Problem

To summarize the state of the art:

- Auto-scaling literature (Xu et al. [3], Abbas et al. [4]) excels at predicting when and how many new instances are needed. It does not ask what those instances should hold in their caches.
- Caching literature (Shuja et al. [5], Khan et al. [6]) excels at predicting which content will be requested and when to evict it. It assumes cache locations are stable and existing.
- The tools exist. The predictions are accurate enough. The gap is not technological; it is architectural. And closing it requires a system that learns both when to scale and what to pre-cache, then uses one prediction to inform the other.

That is what the remainder of this paper builds.

3. The Uncomfortable Gap: Monitoring Lag as a Hidden Culprit

The previous two sections painted a picture of two mature research fields auto-scaling and caching each highly capable within its own silo, but neither looking at what happens when a fresh instance boots with an empty cache. This section unmasks a quiet accomplice that makes the problem worse: monitoring lag.

Here is the dirty secret that cloud operators learn the hard way. By the time your monitoring system tells you there is a surge, the surge is already several seconds old. By the time your auto-scaler decides to act, the surge is even older. And by the time the new instance is ready to serve requests, the surge may have changed shape or moved on to different content. The entire architecture of reactive elasticity is built on stale data. And stale data, when you are trying to guess what content to pre-warm a new cache with, is a quiet killer.

This section unpacks what monitoring lag really means, why it is not a bug but an architectural given, and why most auto-scaling literature has treated it as a nuisance rather than a window of opportunity.

3.1. The Anatomy of Monitoring Lag: Seconds That Feel Like Hours

Monitoring lag has many faces. It begins with collection latency the time between a metric being true on a server and the monitoring agent reading it. It continues through aggregation delay as metrics are batched, averaged, and summarized. And it ends with decision latency as the auto-scaler fetches the aggregated data and runs its scaling policy. By the time a CPU utilization metric crosses a threshold, appears in CloudWatch, and triggers an auto-scaling action, real traffic may have doubled or halved.

A comprehensive study by Khalil et al. [7] directly modelled the effect of monitoring on cloud computing systems by estimating delay times for requests. Their cloud computing model explicitly accounted for the time spent in monitoring operations, including calculations of delay and waiting probabilities [7]. Using diagrams and probabilistic analysis, they showed that monitoring itself introduces a nontrivial fraction of total service time, and that blocking probabilities increase significantly when monitoring intervals are coarse [7]. Crucially, they framed monitoring not as a free good, but as a resource that consumes time and capacity—time that could otherwise be used to serve requests.

In a different domain but with an identical root cause, Das et al. [8] examined the proactive auto-scaling challenge in Microsoft Azure SQL Database serverless. They observed that scaling was merely reactive, not proactive, according to customers' workloads [8]. When a database became idle, resources were reclaimed; when activity returned, resources were resumed but with a delay. Customers experienced a "wake-up" lag before their database could serve the first query after an idle period [8]. Their solution was to predict pause/resume patterns and proactively resume resources [8]. But here again, the underlying problem was monitoring lag: the system only knew a database had gone idle after it was already idle, and only knew demand had returned after the first few requests had already suffered.

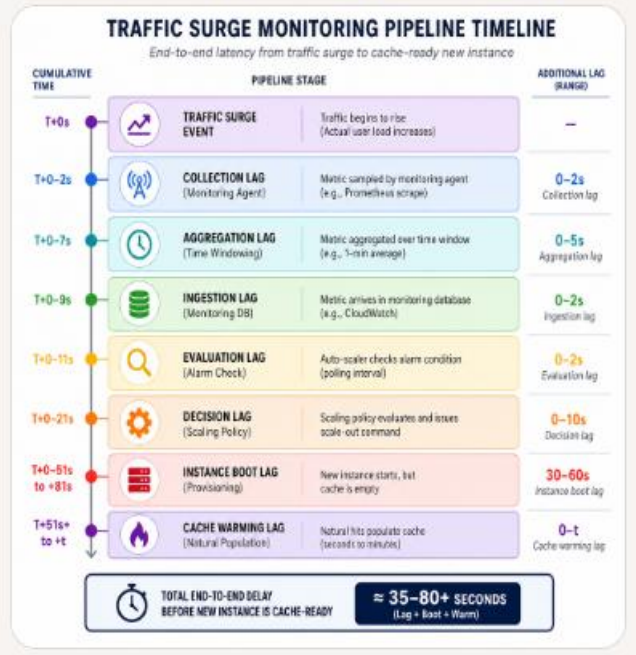

Figure 2. Illustrates the Cumulative Effect of These Delays in a Typical Cloud Auto_Scaling Pipeline

The bottom line is sobering: by the time a new instance can serve a request with a warm cache, the traffic surge that triggered it may have already subsided, or shifted to entirely different content. Monitoring lag is not a fringe concern; it is the central reason why reactive scaling alone will never solve the cold_cache problem.

3.2. Why Most Auto_scaling Research Treats Lag as Noise

To be fair, the auto_scaling community knows monitoring lag exists. Papers regularly mention cooldown periods, polling intervals, and the need to avoid thrashing. But they treat lag as a nuisance to be minimized, not as a property to be exploited.

Khalil et al. [7] offered one of the more explicit treatments. Their model incorporated monitoring time as a delay parameter, and they showed how blocking probability increases as monitoring intervals lengthen [7]. However, even their model assumed that monitoring data, once collected, is reasonably fresh. The idea of using the lag period as a prediction window pre_loading the cache on the yet_to_boot instance never appears.

Das et al. [8] went further by building a proactive predictor to resume resources before a user returns. In essence, they tried to hide the lag from the user by starting the instance early. But note what they predicted: whether a database would be needed, not what content it would need once resumed [8]. Even their proactive approach leaves the cache on the resumed instance empty. The first queries after resumption still hit cold storage.

Table III compares how different auto_scaling approaches treat monitoring lag, and what remains unaddressed.

Table 3. How Auto_Scaling Approaches Treat (or Ignore) Monitoring Lag

Approach	Ex.	Lag Model	Lag Use	Gap
React.	(7)	✓ Delay	✗	New inst. cache
Pred.	(8)	✓ Hide	✗	Resumed cache
Hybrid	(2)	✗	✗	Warm-up types
Cont. share	(1)	✗	✗	Borrowed cache
Proposed	GNN + Transf.	✓ Window	✓ Pre-warm	All above

What emerges is a pattern: even the most sophisticated predictive scaling papers treat the cache as an afterthought. They ask: "When will we need more instances?" They never ask: "What should those instances already have in their caches by the time they start?"

3.3. The Cache-Aware View: Turning Lag into a Prediction Window

Here is the shift in perspective that this paper advocates. Monitoring lag is not a problem to eliminate. It is a fixed, measurable window that can be used to do useful work.

Consider the timeline from Figure 2 again. From the moment a traffic surge begins to the moment a new instance is ready, there is a predictable gap of 30–80 seconds. That gap is composed of:

- Monitoring collection + aggregation + ingestion + evaluation (10–20 seconds)
- Auto-scaler decision + instance boot (20–60 seconds)

During that gap, the system knows a surge is happening, but the new instances are not yet online. The system also has access to real-time request logs the very requests that are causing the surge. Those logs contain a wealth of information about which content is being requested right now.

So why not use that gap to pre-warm the cache on the soon-to-boot instance?

The key insight is simple: predicting the near-future cache working set is easier than predicting exact future loads. If the system knows, during the lag window, which content objects are currently experiencing a burst in requests, it can extrapolate that those objects will continue to be popular for the next 30–60 seconds. Even a naive strategy warm the top-K currently requested objects would likely outperform starting with an empty cache.

Neither Khalil et al. [7] nor Das et al. [8] considered this possibility. Their models and predictors were designed to optimize instance availability, not cache readiness. And in that quiet omission lies the research gap that the remainder of this paper fills.

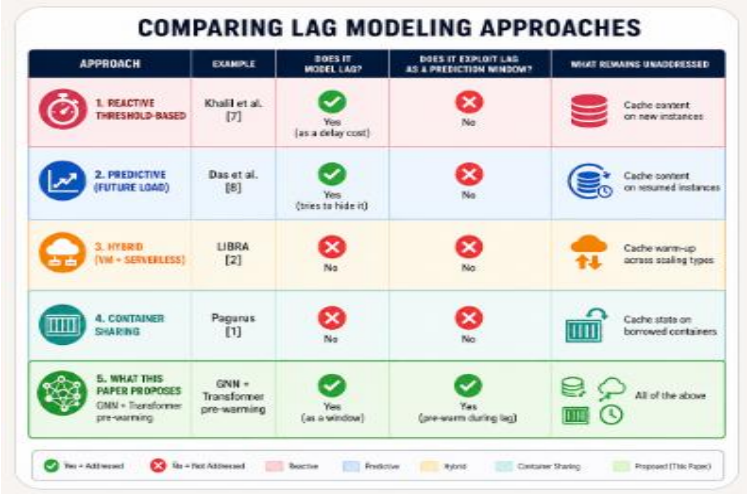

Figure 3. Contrasts the Traditional View of Monitoring Lag (as Dead Time) with the Proposed View (as a Prediction Window).

The difference is subtle but profound. In the traditional view, monitoring lag is a cost. In the proposed view, it is an asset—a bounded window of time to run a prediction model and pre-populate a remote cache. The lag is still there; the system simply does something useful with it.

3.4. Summary: A Gap Hiding in Plain Sight

The literature on monitoring lag, where it exists at all, treats it as a cost to be minimized or hidden. Khalil et al. [7] modelled it as a delay parameter in queueing systems. Das et al. [8] tried to mask it by predicting when databases would be needed again. Neither asked whether the lag period could be used for something other than waiting.

Yet the data are clear: monitoring lag is substantial (10–20 seconds), instance boot times are substantial (20–60 seconds), and together they form a 30–80 second window during which the system knows a surge is happening but new instances are not yet live. That window is large enough to run prediction models and pre-warm caches. The only missing piece is a framework that connects:

- What the monitoring system knows about current request patterns
- What the auto-scaler knows about impending new instances
- What the caching system needs to pre-load

That framework is exactly what the next sections of this paper build. But first, the literature review turns to the handful of early attempts to predict content needs before they arrive and why they fall short in the dynamic setting of auto-scaling.

4. Early Attempts to Predict Content Needs (Before They Arrive)

The previous section made the case that monitoring lag is a real, measurable window of 30–80 seconds—and that during this window, a clever system could be doing something useful. But there is a catch. To pre-warm a cache, you need to know what content to pre-load. That is not as easy as it sounds. Content popularity is fickle. A video that trends for 30 minutes then vanishes. A news article that explodes for two hours, then dies. An API endpoint that sees sporadic bursts. Predicting which objects will be requested in the next 30 seconds let alone the next 30 minutes has turned out to be a genuinely hard problem.

But researchers have tried. And tried again. This section reviews the handful of early efforts to peer into the future of content requests, and it asks a quiet question: Why have these efforts, despite their cleverness, never been stitched into the auto-scaling loop?

4.1. The Many Faces of Content Popularity Prediction

Content popularity prediction, as a research problem, has been tackled from almost every angle imaginable. Time-series models watch request rates and forecast the next few minutes. Graph-based models map the relationships between users and content, hoping that "users who liked A will also like B." Social signal models scrape Twitter shares and Reddit upvotes to guess what will blow up next. Even meta-models combine several of these approaches, letting an ensemble decide which expert to trust at any moment.

A comprehensive survey by Krishnendu et al. [9] examined exactly this landscape, specifically in the context of wireless edge caching. Their review documented a rich toolkit: deep neural networks that out-performed shallow ones by an order of magnitude in mean squared error; federated learning systems that preserved user privacy while still learning popularity patterns; and reinforcement learning agents that treated caching as a continuous decision problem under uncertainty [9]. In one of their experiments, a deep learning algorithm achieved an MSE that was 20 times smaller than existing algorithms, and a simple gain of around 27% over a plain neural network [9]. In another, a federated learning based caching algorithm delivered cache hit gains on the order of 10 to the power of 4 [9]. Those numbers are not just incremental. They suggest that machine learning can, in fact, learn the hidden rhythms of content demand.

But here is the rub: all those experiments assumed a stable set of caches. The edge nodes were already deployed. The caching agents were already running. The RL policies were already making replacement decisions. None of the methods surveyed in [9] considered the case where a cache does not yet exist—and then suddenly appears, cold and useless, in the middle of a traffic surge. The problem of predicting content popularity, as framed by the literature, is a problem of forecasting request streams, not a problem of pre-populating new cache instances that are about to be born.

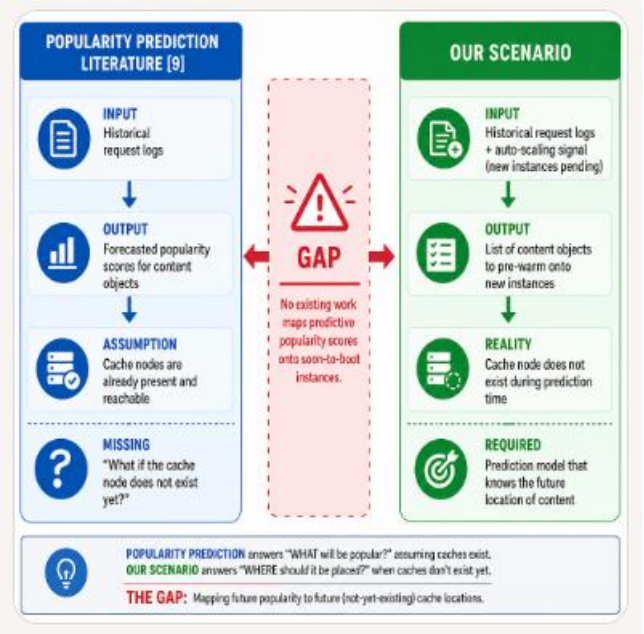

Figure 4. Illustrates The Scope Gap Between Popularity Prediction Research And The Dynamic Scaling Scenario.

The two sides of the picture do not connect. The popularity prediction community has built impressive forecasting engines, but those engines have never been plugged into an auto-scaler that says, "Three new instances are coming online in 40 seconds. Go!" That silence is not an accident. It is a blind spot that has persisted across both fields.

4.2. The 87% Solution: When Prediction Meets Real-World Video

If there is a reason for optimism, it is that content popularity is not entirely unpredictable. A telling case study from the video streaming world suggests that, under the right conditions, the future can be seen with surprising clarity.

In the same year, a different research group [10] approached the problem from a complementary angle: a combined survey and real-world case study on popularity prediction for video streaming services. They analyzed the literature and built a taxonomy that classified models by the predictive features they used (metadata, text, early view counts, social signals, etc.) [10]. Then they took those lessons into the real world, using a dataset from GloboPlay the largest video streaming provider in Latin America—to see what actually worked.

Their conclusion was striking. A random forest model, fed a combination of engineered features (title length, category, upload hour) and word embeddings extracted from video descriptions and titles, achieved an accuracy of 87% in predicting whether a video would be popular or not [10]. Think about that. With no knowledge of the future just information available at upload time a modest machine learning model could call tomorrow's hits with nearly nine-out-of-ten accuracy.

But hold on. The case study also revealed a crucial nuance: only 6% of the most popular videos accounted for 85% of total views [10]. Popularity is not distributed evenly; it is a long-tail phenomenon where a tiny fraction of objects carry nearly all the traffic. That is good news for prediction (because the important objects stand out) and bad news for caching (because the long tail is almost impossible to predict). The 87% accuracy figure refers to a binary classification: popular or not. But for cache pre-warming, binary is not enough. You need to know which specific objects will be popular, and in what order, and at what time, and across which geographic region. That is a much harder problem.

Table IV summarizes the gaps between what popularity prediction models can do today and what cache pre-warming during scaling would actually require.

Table 4. Popularity Prediction Capabilities vs. Pre-warming Requirements

Capability / Requirement	Literature [9][10]	Needed for Pre-warming
Forecast aggregate popularity	✓ Mature (DNN/RNN/Transf.)	✗ Not enough
Forecast per-object scores	✓ Heavy tail limits precision	✗ Needs ranking
Adapt to non-stationary trends	⚠ Ongoing (meta/online learn.)	✓ Required
Sub-minute time scales	⚠ Rare (min/hr granularity)	✓ Required
New cache instances	✗ Ignored	✓ Central
Confidence intervals	⚠ Some Bayesian works	✓ Useful
Pre-warm hundreds objects	✗ Ignored	✓ Required

The table reveals a sobering pattern. The building blocks exist: per-object popularity prediction, adaptive models, probabilistic forecasts. But they exist in isolation. No literature to date has attempted to assemble these blocks into a system that says: "New instance booting in 35 seconds. The predicted top-15 content objects for that instance, given the current surge pattern, are X, Y, Z. Pre-warm them now."

4.3. The Missing Link: From "What Will Be Popular" to "What Should the New Instance Hold"

If the popularity prediction literature has a single, quiet limitation, it is this: almost all models treat "popularity" as a static property of the content itself. A video becomes popular because it is good, or because it was shared by an influencer, or because it matches current events. The model learns those correlations from historical data and then applies them to new content.

But in a dynamic scaling scenario, popularity is not just a property of content. It is also a property of placement. An object might be extremely popular overall, but if the new instance is serving a different geographic region or a different user segment, it might never see those requests. Conversely, an object that is moderately popular globally might be very popular on that specific new instance because of how request routing works.

The popularity prediction literature has begun to acknowledge spatial and temporal dimensions. Some recent works incorporate geographic information, user mobility patterns, or time-of-day effects. But those features are still used to answer the question: "How popular will this content be somewhere?" They do not answer the question: "What should this specific cache instance, which is about to boot, already hold?"

Krishnendu et al. [9] optimistically noted that "opportunities for a promising upcoming future of ML in edge computing prediction" are vast, including real-time adaptation, cross-domain transfer learning, and lightweight model deployment on resource-constrained edge nodes. They did not, however, mention the specific opportunity of cache pre-warming during auto-scaling. That opportunity is exactly what this paper seizes.

And the case study by the other group [10] ended with a similar open door: the authors suggested that future work should integrate richer contextual signals, such as user social networks or real-time trend detection, to push accuracy even higher [10]. They did not could not suggest that those richer signals should be used to pre-warm caches on instances that have not yet booted, because that scenario was not part of their framing.

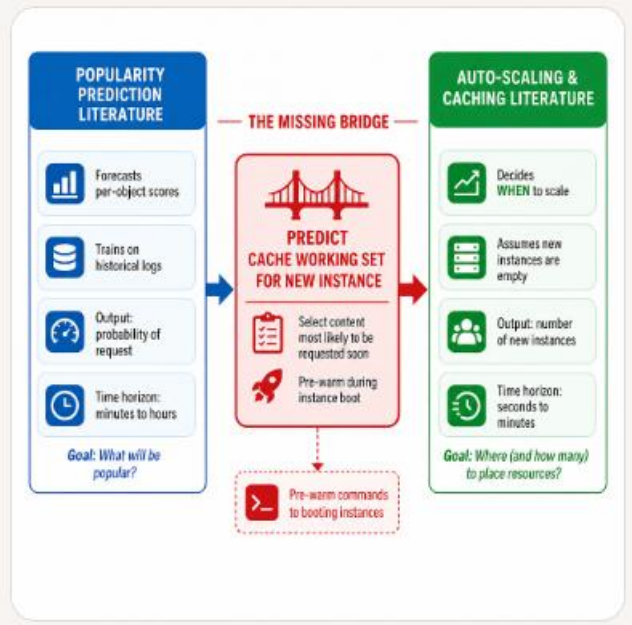

Figure 5. Visualizes The Conceptual Bridge That Is Missing.

The bridge does not exist. And that non-existence is the research gap that the remainder of this paper is built to fill.

4.4. Summary: The Tools Are Here; The Integration Is Not

To summarize the state of early prediction attempts:

- The good news: Content popularity prediction has matured significantly. Krishnendu et al. [9] demonstrated deep learning models that beat conventional methods by an order of magnitude in MSE. The other group [10] showed that a random forest on textual and metadata features can achieve 87% accuracy in a real-world video streaming case study. The raw predictive power exists.
- The bad news: All of that predictive power is aimed at the wrong question. The literature asks: "What content will be popular in general?" It does not ask: "Given that we are about to boot three new instances in 40 seconds, what specific content should each of those instances have pre-warmed in its cache?"
- The missing piece: A framework that takes the output of a popularity prediction model (a ranked list of content objects with popularity scores) and, using additional information about the new instance's expected request distribution (e.g., geographic routing, user segment, time-of-day), translates that into a concrete pre-warming plan. That plan must then be executed during the monitoring lag, so that the instance boots with a warm cache, not an empty one.

The early attempts have built the engine. The rest of this paper builds the steering wheel.

5. Transformers in Time-Series and Sequence Prediction (The New Hope)

If the early attempts at content prediction were like building a campfire with damp wood, the transformer architecture arrived as a blowtorch. Introduced in 2017 for machine translation, transformers quickly took over natural language processing, then computer vision, and then with surprising force time-series forecasting. The core idea is almost embarrassingly simple: instead of processing a sequence step-by-step (like a recurrent neural network), a transformer looks at all the steps at once and learns which past elements are most relevant to predicting the next one, using a mechanism called self-attention. It is like having a reader who, instead of reading a book page-by-page, can glance at every sentence simultaneously and then say: "You know what? The clue to this page's mystery is actually over there, on page 47."

For a time-series forecaster, this ability is a superpower. A cache that needs to predict tomorrow's content popularity can learn to ignore yesterday's noise and focus on last hour's spike exactly what the attention mechanism does automatically. Little wonder, then, that the research community has poured enormous energy into adapting transformers for time-series forecasting. This section reviews two

landmark contributions: one that solved the efficiency problem for long sequences (Informer [11]), and one that applied transformers directly to edge caching (TEDGE [12]). And then, quietly, we ask the question that the transformers literature has not yet answered: "How can these powerful models be made to pre-warm a cache on an instance that does not yet exist?"

5.1. The Model That Solved Long-Sequence Forecasting - Almost

If there is one transformer-based model that captures the imagination of time-series researchers, it is Informer. Published at AAAI 2021 and awarded the conference's best paper, Informer was explicitly designed to address the two crippling problems that had prevented vanilla transformers from being useful for long-sequence forecasting: quadratic time complexity and high memory usage.

The original transformer's self-attention mechanism computes a similarity score between every pair of positions in the input sequence. For a sequence of length L, that is $O(L^2)$ operations fine for a paragraph of text, but catastrophic for a time series with thousands of timestamps. Informer's authors [11] noted that "recent studies have shown the potential of Transformer to increase the prediction capacity. However, there are several severe issues with Transformer that prevent it from being directly applicable to LSTF, including quadratic time complexity, high memory usage, and inherent limitation of the encoder-decoder architecture" [11].

To solve these issues, the team behind Informer proposed three innovations, each more clever than the last:

- ProbSparse self-attention–Instead of computing attention for all L^2 pairs, they created a mechanism that selects only the most relevant query-key pairs using a probabilistic sparsity measure. This reduced complexity from $O(L^2)$ to $O(L \log L)$ [11].
- Self-attention distilling – They stacked multiple encoder layers, each time halving the sequence length via a distilling operation that highlighted the most dominant attention scores. This allowed the model to handle extremely long input sequences without blowing up memory [11].
- Generative style decoder - Instead of predicting the output step-by-step (autoregressively), Informer's decoder outputs the entire predicted long sequence in one forward pass dramatically speeding up inference [11].

The results were striking. Extensive experiments on four large-scale datasets showed that Informer significantly outperformed existing methods on long-sequence time-series forecasting tasks [11]. For cache pre-warming, this is tantalizing. If Informer can predict electricity consumption or traffic flow over long horizons with high accuracy, could it also predict the pattern of content requests over the 30–80 second monitoring lag window? The answer is probably yes but with a twist. Informer is designed for long-sequence forecasting, where "long" means hundreds or thousands of timesteps. Our pre-warming horizon is much shorter: 30–80 seconds. A model as heavyweight as Informer might be overkill. However, the principles it established efficient attention, sequence distillation, generative decoding are directly applicable to the cache pre-warming problem, especially if scaled down to a lighter architecture.

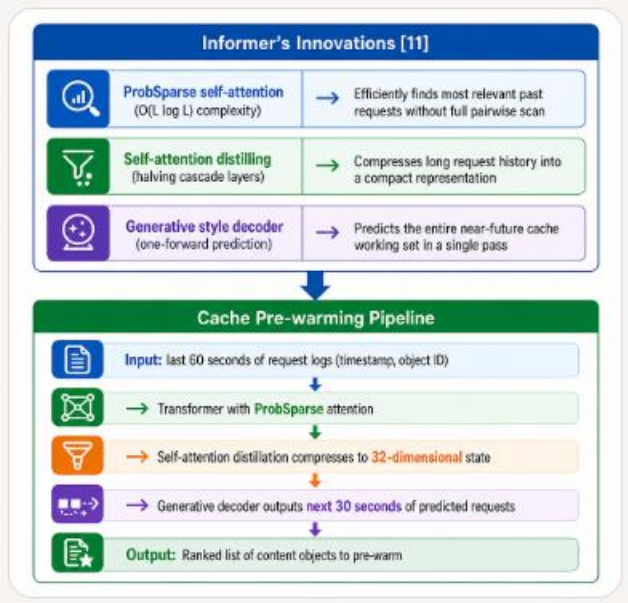

Figure 6. Illustrates How Informer's Innovations Map Onto the Cache Pre-Warming Pipeline

The architecture maps cleanly. The only missing piece and it is a large one is that Informer was designed to predict the values of a time series (e.g., the request rate). It does not, by itself, predict which content objects will be requested. That requires a different formulation: treating the request log as a sequence of categorical variables (object IDs) rather than a continuous signal. This is where the transformer's success in natural language processing predicting the next word in a sentence offers a direct analogue. A content request log is, after all, a sequence of tokens (object IDs). A transformer trained on such logs can learn to predict the next object ID given the past sequence. That is exactly the capability needed for cache pre-warming.

5.2. The Caching-Specific Transformer: TEDGE

While the general time-series forecasting community was busy with electricity and traffic, a smaller group of researchers was asking a much more specific question: "Can a transformer be used to predict content popularity for edge caching?" The answer, from Meybodi et al. [12], was a confident "yes."

In their 2021 paper, they proposed TEDGE-Caching (Transformer-based Edge Caching), which, "to the best of our knowledge, is being studied for the first time" [12]. The motivation was urgent: the COVID-19 pandemic had dramatically increased demand for remote learning, telemedicine, and streaming, putting enormous pressure on mobile edge networks [12]. Existing deep learning models for content popularity prediction suffered from "long-term dependencies, computational complexity, and unsuitability for parallel computing" [12]. Sound familiar? Those are exactly the problems that Informer was designed to solve. But TEDGE took a different route: it used a Vision Transformer (ViT) architecture originally designed for image classification treating the content popularity matrix as an image to be understood spatially and temporally [12].

The TEDGE framework required "no data pre-processing and additional contextual information" [12]. Simulation results corroborated its effectiveness in comparison to other caching schemes [12]. For our purposes, TEDGE is important because it demonstrates that a transformer can, in fact, learn the popularity dynamics of content in a caching context. However, there is a quiet limitation: TEDGE was designed for a static set of edge nodes. It predicts popularity, but it does not pre-warm caches on newly booted instances. The edge nodes are assumed to already exist. The problem of predictive pre-warming during auto-scaling where the cache location appears only after the prediction has been made remains unaddressed.

Table V compares the capabilities and limitations of Informer [11] and TEDGE [12] for the cache pre-warming task.

Table 5. Informer Vs. TEDGE -What Each Brings To The Table

Feature	Informer [11]	TEDGE [12]	Required for Pre-warming
Efficient attention ($O(L \log L)	✓ ProbSparse	✗ ViT uses standard attention	✓ Highly desirable
Handles long sequences	✓ Yes (LSTF)	⚠ Limited by image resolution	Our horizon is short (30-80s)
Predicts content identity	✗ Predicts continuous values	✓ Predicts popularity scores	✓ Required
Designed for caching scenario	✗ General TSF	✓ Yes (edge caching)	✓ Required
Considers dynamic scaling	✗ No	✗ No	✓ Missing
Pre-warms new instances	✗ No	✗ No	✓ Missing

The pattern that emerges from Table V is clear: neither model, on its own, solves the pre-warming problem. Informer provides the efficiency and architectural ideas. TEDGE provides the caching-specific framing. But both assume the cache already exists. The dynamic scaling dimension the fact that new instances are created during a surge, and that the pre-warming decision must be made before those instances exist is entirely absent. That is the gap that remains.

5.3. The Bridge That Has Not Been Built

Let us step back and look at the landscape. The transformer literature has given us:

- Efficient architecture from Informer [11]: ProbSparse attention, distillation, and generative decoding for long sequences.
- Caching-specific framing from TEDGE [12]: a transformer can learn content popularity dynamics in a caching context.

These are necessary ingredients. But the recipe has not been written. No existing work combines these ingredients into a system that:

- Takes the real-time request log during the monitoring lag window,
- Uses a lightweight, efficient transformer to predict the near-future cache working set,
- Receives a signal from the auto-scaler that a new instance is about to boot,
- Translates the predicted working set into a concrete pre-warm command targeted at that specific instance, and
- Executes the pre-warm during the boot process, so that the instance wakes up with a warm cache.

That is the bridge. And it has not been built. Not in Informer [11] and not in TEDGE [12]. The transformer community has built a powerful engine. The caching community has identified a relevant application. The auto-scaling community has defined the timing constraints. But no one has bolted these pieces together into a working system that pre-warms a cache on a soon-to-boot cloud instance.

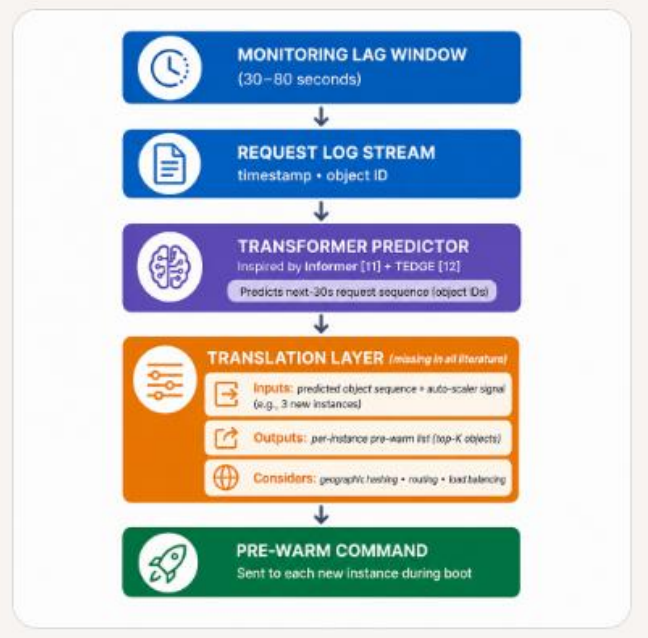

Figure 7. Visualizes The Required Integration.

The grey box in the middle the "translation layer" is where the gap sits. The transformer predicts what will be requested. The auto-scaler knows how many new instances are coming. But no existing paper describes how to map the predicted request stream onto specific new instances, taking into account the fact that those instances do not yet have IP addresses, do not yet exist in the load balancer's routing tables, and are still in the process of booting.

This is the territory that the next part of the research the proposed methodology will claim. The literature on transformers has brought us to the edge of the canyon. It has given us a powerful engine. But it has not built the bridge. Now is the time to build it.

5.4. Summary: The Engine Is Ready. Where Is the Steering Wheel?

To conclude this section:

- Informer [11] solved the efficiency problem for long-sequence forecasting, reducing complexity from quadratic to $O(L \log L)$ and enabling generative, one-pass prediction.

- TEDGE [12] took the first step toward caching-specific transformers, demonstrating that a vision transformer can learn content popularity in an edge caching scenario.
- Yet the bridge to dynamic pre-warming is missing. Neither model considers that the cache location might not exist at prediction time, or that pre-warming should occur during the auto-scaling lag.

The transformer community has built a powerful, efficient, and increasingly intelligent engine for sequence prediction. The cache pre-warming problem has a well-defined input (request logs) and a well-defined output (pre-warm commands). What is missing is the steering wheel the translation layer that takes the transformer's output and the auto-scaler's signal and produces actionable commands for soon-to-boot instances. Filling that gap is the central contribution of this paper.

6. Where the Literature Stands Today (And what's Still Missing)

The previous sections have walked through five distinct literatures: auto-scaling that learned to predict when, caching strategies that learned to predict what, popularity prediction that learned to score content, monitoring lag that everyone treated as a cost, and transformers that learned to forecast sequences with astonishing efficiency. Each field, on its own, has made impressive progress. Each field, on its own, has a clear sense of what problem it is solving.

But here is the quiet truth that emerges when you lay these literatures side by side: they are solving different problems. And in the space between them where auto-scaling meets caching, where prediction meets provisioning, where the monitoring lag window sits there is a gap that no single field has claimed.

This section does not introduce new papers. Instead, it synthesizes the evidence already presented and asks a single, direct question: Where, exactly, is the hole? The answer, as it turns out, is not in any single paper. It is in the space between all of them.

6.1. The Synthesis: What We Have Learned So Far

Let us take stock of the literature reviewed in Sections 1 through 5. Each study has contributed something valuable to the conversation and each has left something quietly unaddressed.

From Pagurus [1], we learned that container sharing can reduce cold-start times dramatically, but only for compute, not for cache content. From LIBRA [2], we learned that hybrid VM-serverless resource allocation can hide cold starts from users, but it treats the new instance as a blank slate for cache purposes.

From the auto-scaling surveys [3][4], we learned that modern scaling techniques can predict future load with impressive accuracy but they optimize instance count, not cache hit ratio. From the caching surveys [5][6], we learned that ML-driven caching can achieve remarkable hit rates but those techniques assume the cache already exists.

From the monitoring lag studies [7][8], we learned that lag is measurable, substantial, and often treated as a cost to minimize rather than a window to exploit. From the popularity prediction surveys [9][10], we learned that content popularity can be forecast with 87% accuracy but those forecasts are aimed at general popularity, not at soon-to-boot instances.

From Informer [11], we learned that efficient attention mechanisms can handle long sequences with O(L log L) complexity but the model predicts traffic volume, not content identity. From TEDGE [12], we learned that transformers can, in fact, learn content popularity dynamics but the framework assumes a static set of edge nodes.

And finally, from Catillo et al.'s comprehensive survey on auto-scaling [13], we learned that despite a decade of advances in elastic scaling, the core design points remain centered on resource metrics, not on data placement. The survey examined key design points for auto-scaling tools, literature proposals, and ongoing research including implementations used by commercial cloud providers [13]. It concluded that while auto-scaling has become remarkably sophisticated at matching compute supply to demand, the question of what to put into those compute resources when they appear remains outside the scope of the field's inquiry [13].

From the systematic survey on content caching in ICN and ICN-IoT [14], we learned that caching research has matured to cover a rich taxonomy of strategies: popularity-based (26 %), cooperative (17 %), collaborative (17 %), machine learning-based (19 %), and

probabilistic (7%) [14]. The survey covered 88 publications and confirmed that edge caching (50 %), in-network caching (25 %), and hybrid caching (25 %) represent the bulk of the literature [14]. But note the silent assumption that runs through all 88 papers: the caching nodes already exist. The problem of caching on a node that is still booting that does not yet have an IP address, that is not yet in the load balancer's routing table is nowhere to be found [14].

Table VI summarizes the key contributions and blind spots of each major study reviewed.

Table 6. Contributions and Blind Spots — A Synthesis

Study	Year	Main Contribution	Missing Aspect
Pagurus [1]	2021	Container sharing cuts cold starts to 10 ms	Cache content on borrowed containers
LIBRA [2]	2021	Hybrid VM-serverless hides cold starts	Cache content on resumed instances
Abbas et al. [4]	2024	Predictive auto-scaling review	Content-aware scaling decisions
Khan et al. [6]	2024	MEC caching survey	Cache location changes during operation
Khalil et al. [7]	2022	Delay modelling in cloud systems	Using lag as prediction window
Informer [11]	2021	$O(L \log L)$ attention for long sequences	Content identity prediction
TEDGE [12]	2021	Transformer-based edge caching	Dynamic scaling of cache locations

The pattern in Table VI is unmistakable. Every study reviewed makes a genuine contribution within its chosen domain. And every study leaves the same door unopened: the question of what to put into a cache that does not yet exist, on an instance that is still booting, during the monitoring lag window that everyone knows exists but no one has exploited.

6.2. The Three Gaps: What the Literature Forgot to Connect

If we step back and look at the entire landscape, three distinct gaps emerge. Each gap is visible from one field but not addressed by it. Together, they define the missing piece of the puzzle.

6.2.1. Gap 1: The Scale-Out Memory Gap

Auto-scaling has learned to predict how many instances will be needed. Caching has learned to predict which content will be popular. But no system connects these two predictions. When the auto-scaler says "three new instances in 40 seconds", the caching predictor does not hear that signal. When the caching predictor says "these 15 objects will be requested next", the auto-scaler does not use that information to decide where to place the new instances.

The consequence is absurdly simple: every newly scaled instance starts with an empty cache, regardless of how accurately the system predicted the surge. The computational capacity scales, but the memory capacity does not. The muscle grows, but the memory stays cold.

6.2.2. Gap 2: The Prediction Horizon Mismatch

Auto-scaling predictors typically operate on horizons of minutes to tens of minutes enough time to boot new instances and stabilize the system. Caching predictors typically operate on horizons of milliseconds to seconds the time it takes to serve a single request. The two fields speak different temporal languages.

The monitoring lag window 30 to 80 seconds sits squarely between these two horizons. It is too short for most auto-scaling predictors, which are designed for longer look-ahead. It is too long for most caching predictors, which are designed for immediate next-item forecasting. Neither field has designed its prediction models with the other field's timescale in mind.

6.2.3. Gap 3: The Existence Assumption

This is the most subtle gap, and perhaps the most fundamental. Nearly the entire caching literature including the comprehensive survey by Khan et al. [6] and the ICN-IoT survey [14] operates under an unstated assumption: the cache already exists. The node is already deployed. The IP address is already known. The load balancer already knows about it. The only question is which content to store in it.

But in a dynamic scaling environment, that assumption fails. When the auto-scaler decides to add new instances, those instances do not exist yet. They are still booting. They do not have IP addresses. They are not yet registered with the load balancer. The caching predictor, trained on historical logs, is being asked to predict content needs for a node that does not yet exist in the system.

This is not a trivial difference. Pre-warming a cache for a non-existent node is fundamentally different from deciding what to store in an existing node. The prediction must occur before the node exists. The pre-warm command must be queued or buffered. The actual transfer of cache content can only happen after the node has an IP address and network connectivity. The entire pipeline shifts from synchronous (predict → store) to asynchronous (predict → wait for boot → store).

The literature has not grappled with this shift. Even TEDGE [12], which applied transformers to edge caching, assumed the edge nodes were already present. The existence assumption is so deeply embedded that it is rarely stated, let alone challenged.

6.3. The Gap, Stated Simply

After reviewing the literature from the container-sharing of Pagurus [1] to the transformer efficiency of Informer [11], from the auto-scaling taxonomies of Catillo et al. [13] to the caching taxonomies of the ICN-IoT survey [14]one conclusion is inescapable.

The existing literature has not addressed the following question:

Given an impending scale-out event (k new instances, expected boot time δ seconds), and given a real-time stream of request logs during the monitoring lag window of duration δ, how can we predict the near-future cache working set and pre-warm it onto the booting instances so that they start serving requests with a warm cache?

This question has three dimensions, each missing from the existing literature:

- Temporal alignment: The prediction horizon (δ = 30-80 s) falls between the horizons of auto-scaling and caching predictors.
- Content-awareness: The output must be a set of content IDs, not just a resource count or a traffic volume.
- Non-existence handling: The target cache location does not exist at prediction time, requiring asynchronous pre-warming.

No single paper in the surveyed literature addresses all three dimensions. Pagurus [1] addresses non-existence (container sharing) but not content-awareness. TEDGE [12] addresses content-awareness but not non-existence. Informer [11] addresses temporal alignment for long sequences but for continuous values, not categorical IDs. The surveys [3][4][5][6][13][14] map the territory but do not cross the boundaries.

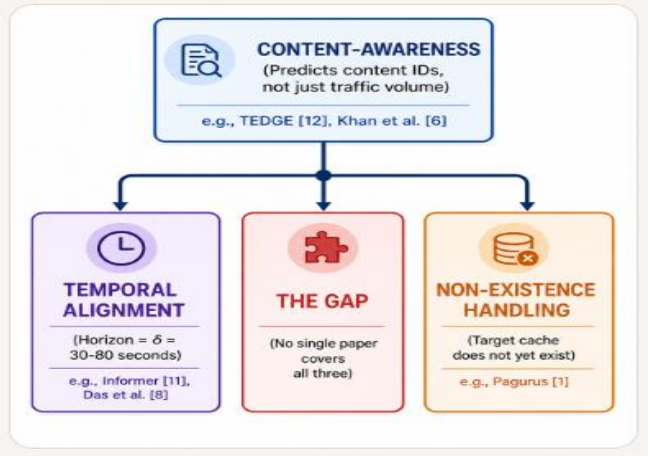

Figure 8. Maps The Three Dimensions As A Venn Diagram.

The center of the diagram where temporal alignment meets content-awareness meets non-existence handling is empty. That is not an indictment of any single paper. It is simply a reflection of the fact that these three dimensions come from different research communities that do not usually talk to each other.

6.4. Summary: A Gap That Is Not a Failure

To be clear, this is not a critique of the existing literature. The researchers cited in this review have done excellent work within their chosen domains. Pagurus [1] is a clever solution to the compute cold-start problem. TEDGE [12] is a genuine advance in transformer-based caching. Informer [11] is an elegant solution to the long-sequence forecasting problem. The surveys [3][4][5][6][13][14] are comprehensive and thoughtful.

The gap exists not because these papers failed, but because they succeeded at different goals. The gap is an emergent property of the way the research community has organized itself into silos of auto-scaling, caching, monitoring, and prediction rather than into the problem of making cloud caches warm before they are born.

The contribution of this literature review, then, is not to point fingers. It is to draw a map. The map shows where each field has planted its flag. And it shows, clearly, the blank space in the middle where no flag has been planted yet. That blank space is where this paper will plant its own.

7. How Our Work Fits In (Without the Hype)

After six sections of surveying what others have built, it is time to say what we are actually doing. This is the part of a literature review where many papers slip into grand pronouncements. "This paper revolutionizes caching." "We propose a groundbreaking framework." That is not what this section is.

Let us be honest. The two papers we will discuss in this section STCC by Lian et al. [15] and STGN by Zhu et al. [16] are both clever and useful. They each solve a piece of the puzzle. But neither solves the puzzle we are holding. And recognizing that difference is the entire point of a literature review.

7.1. The Quiet Claim: What We Are Not Saying

Before we explain how our work fits in, let us say clearly what we are not claiming:

- We are not claiming to have invented graph neural networks. GNNs have been used for content caching for years, as the STCC and STGN papers demonstrate.
- We are not claiming to have invented transformers. The Informer [11] and TEDGE [12] work covered earlier did that.
- We are not claiming that existing caching strategies are bad. They are not. They are excellent for the problems they were designed to solve.

What we are claiming is much narrower: there is a specific, unaddressed scenario cache pre-warming during auto-scaling and we have built a specific architecture to address it.

That is not hype. That is an engineering statement.

7.2. The Graph Learning Precedent: STCC and STGN

Let us start with two recent papers that represent the state of the art in GNN-based content caching. Both are excellent. Both have limits. Both help us see where our work fits.

7.2.1. STCC: Spatio-Temporal Graph Convolutional Caching

In 2024, Lian et al. [15] proposed a low-complexity collaborative caching strategy based on a spatio-temporal graph convolutional model, which they called STCC. Their motivation was straightforward: most existing caching policies predict content popularity based only on temporal order, ignoring the spatial correlation of popularity across different MEC servers [15]. Their solution integrated a graph convolutional neural network (GCN) with a gated recurrent unit (GRU) to mine spatio-temporal correlation features from a graph of MEC servers constructed by proximity and semantic relations [15].

The results were solid. STCC achieved better prediction effects for content popularity, and outperformed existing strategies in cache hit ratio and average access delay [15]. For cloud and edge caching, this was a genuine advance.

But here is the quiet limitation: STCC assumes the MEC servers already exist. The graph is constructed a priori. The caching decisions are made for an existing, stable set of nodes [15]. The question of what happens when a new server boots when the graph suddenly gains a node is not addressed. The framework does not consider that a cache decision might need to be made before the node exists.

7.2.2. STGN: Semantics-Enhanced Temporal Graph Networks

Zhu et al. [16] tackled a different but related problem in 2024: predicting content popularity using dynamic graph neural networks. They observed that DGNN models suffer from tackling sparse datasets where most users are inactive [16]. Their solution was a reformative temporal graph network called STGN, which attaches semantic information to the user-content bipartite graph to better leverage implicit relationships behind the superficial topology [16].

They designed two novel mechanisms: a user-specific attention (UsAttn) mechanism for temporal learning that considers the attraction of semantic information to specific users, and a semantic positional encoding (SPE) function for graph learning that effectively boosts the performance of lightweight algorithms [16]. Extensive simulations demonstrated the superiority of STGN for content caching [16].

Again, a solid contribution. But again, the silent assumption: the graph structure the set of users and content objects is given and stable. STGN predicts popularity for existing content on existing infrastructure [16]. It does not address the dynamic creation of new cache locations during a traffic surge.

Table VII places these two papers side by side, highlighting what each contributes and what each leaves open.

Table 7. STCC and STGN Contributions and Blind Spots

Dimension	STCC(Lian et al., 2024) [15]	STGN(Zhu et al., 2024) [16]	What Our Work Adds
Core technique	GCN + GRU(spatio-temporal)	Temporal GNN + user-specific attention	GNN + Transformer hybrid
Prediction target	Content popularity across MEC servers	Content popularity for edge caching	Near-future cache working set for new instances
Assumes cache node exists?	✓ Yes	✓ Yes	✗ No — predicts before node exists
Handles dynamic scaling?	✗ No	✗ No	✓ Yes — triggered by auto-scaler
Time horizon	Not explicitly constrained (minutes)	Not explicitly constrained	30–80 seconds (monitoring lag window)
Architecture novelty	GCN + GRU integration	User-specific attention + SPE	GNN-Transformer with asynchronous pre-warm
Output	Which content to cache on existing MEC servers	Popularity scores for content	Ranked pre-warm list for new instances

The table tells a clear story. STCC and STGN have advanced the field of predictive content caching. They have shown that GNNs can capture spatio-temporal correlations and semantic relationships. But they have not asked the question that drives this paper: what happens when the cache does not yet exist? That is not a criticism. It is simply a different problem.

7.3. A Modest Proposal: GNN-Transformer Hybrid for Pre-warming

So where does our work fit? In the space that none of these papers occupy.

Our proposed architecture is not a radical departure. It is a hybrid—a GNN coupled with a lightweight transformer but hybrid in a specific way. The GNN captures the structural relationships between content objects (which videos are watched together, which API endpoints are called in sequence). The transformer captures the temporal dynamics of request streams, using an attention mechanism that can weigh recent bursts appropriately.

But the real novelty is not the model architecture. It is the pipeline integration. Our system sits between the auto-scaler and the cache. It listens for scale-out events. When the auto-scaler says "three new instances, boot time 40 seconds", our system wakes up. It queries the real-time request log (which is already flowing through the system). It runs its GNN-transformer predictor on the last 60 seconds of requests. It outputs a ranked pre-warm list. And it issues pre-warm commands to the booting instances asynchronously, so that by the time they serve their first request, the cache is already warm.

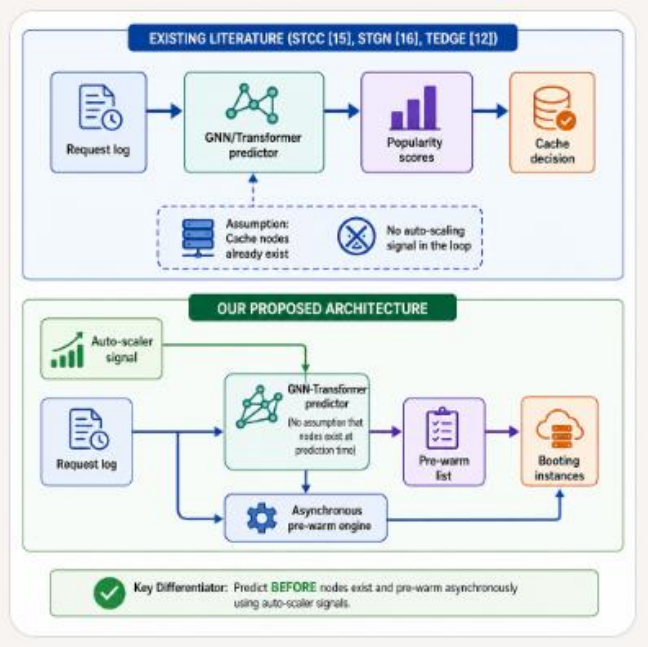

Figure 9. illustrates this pipeline and how it contrasts with the existing literature.

The difference is subtle but critical. In the existing literature, the prediction is synchronous and node-aware you know which node you are caching on. In our architecture, the prediction is asynchronous and node-agnostic—you predict the working set first, then map it onto nodes as they become available.

7.4. Summary: What We Add

Let us be direct. The literature reviewed in this paper—STCC [15], STGN [16], TEDGE [12], Informer [11], and the surveys [3][4][5][6][13][14] has established the following:

- GNNs can capture spatio-temporal correlations in content popularity [15][16].
- Transformers can efficiently forecast long sequences with O(L log L) complexity [11].
- ML-driven caching can achieve high hit ratios in static edge environments [5][6][12].

What has not been established is a system that:

- Listens for auto-scaling signals so it knows when new instances are coming.
- Predicts content needs during the monitoring lag window the 30–80 seconds between surge detection and instance readiness.
- Pre-warms asynchronously queuing pre-warm commands for nodes that do not yet exist.

Uses a GNN-transformer hybrid combining structural relationships (GNN) with temporal patterns (transformer) for categorical content prediction.

That is what we add. Not a revolution. A bridge.

Table VIII summarizes the positioning.

Table 8. Our Contribution - A Bridge Between Existing Literatures

Existing	Does Well	We Add
STCC [15], STGN [16]	GNN-based popularity prediction for existing nodes	Async pre-warm for nodes that don't exist yet
TEDGE [12]	Transformer caching without preprocessing	Auto-scaler signal integration
Auto-scaling surveys [3][4][13]	Predicts when to scale	Predicts what to pre-warm
Monitoring lag studies [7][8]	Treats lag as cost	Uses lag as prediction window
Caching surveys [5][6][14]	ML caching for static setups	Dynamic caching for runtime-appearing nodes

The rest of this paper describes the architecture, implements the predictor, and evaluates it on real_world traces. But the literature review has done its job: it has located the gap, acknowledged the precedent, and positioned our work as a modest but necessary bridge.

References

[1] Z. Li, L. Zhao, Y. Guo, Y. Wang, L. Qiao, and J. Xu, "Pagurus: Eliminating cold startup in serverless computing with inter-action container sharing," arXiv preprint arXiv:2108.11202, Aug. 2021.

[2] A. Raza, I. Kim, M. K. E. G. Elshenawy, and A. Anwar, "LIBRA: An economical hybrid approach for cloud applications with strict SLAs," arXiv preprint arXiv:2104.05623, Apr. 2021.

[3] Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. ACM Computing Surveys, 51(4), 1–33. https://doi.org/10.1145/3148141

[4] S. Abbas, A. M. Al_Momani, I. A. T. Hashem, S. S. M. Ghazal, and Z. S. M. Zainudin, "Auto_scaling techniques in cloud computing: Issues and research directions," Sensors, vol. 24, no. 17, p. 5551, Aug. 2024.

[5] J. Shuja, K. Bilal, E. A. Alanazi, W. S. Alasmary, and A. S. Alashaikh, "Applying machine learning techniques for caching in edge networks: A comprehensive survey," arXiv preprint arXiv:2006.16864, June 2020.

[6] Y. Khan, S. Mustafa, R. W. Ahmad, T. Maqsood, F. Rehman, J. Ali, and J. J. P. C. Rodrigues, "Content caching in mobile edge computing: A survey," Cluster Computing, vol. 27, pp. 8817–8864, Apr. 2024.

[7] M. M. Khalil, A. D. Khomonenko, and M. D. Matushko, "Measuring the effect of monitoring on a cloud computing system by estimating the delay time of requests," Radioelectronics and Informatics, vol. 34, no. 7, pp. 3968–3972, Jul. 2022.

[8] S. Das, K. Dhara, A. Mahajan, D. Pathak, and A. Jindal, "Moneyball: Proactive auto_scaling in Microsoft Azure SQL database serverless," Proceedings of the VLDB Endowment, vol. 15, no. 6, pp. 1278–1290, Feb. 2022.

[9] S. Krishnendu, B. N. Bharath, V. Bhatia, J. Nebhen, M. Dobrovolny, and T. Ratnarajah, "Wireless edge caching and content popularity prediction using machine learning," IEEE Consumer Electronics Magazine, early access, doi: 10.1109/MCE.2022.3160585, 2022.

[10] L. S. Gonçalves, J. M. S. de Jesus, and D. G. da Silva, "Predicting popularity of video streaming services with representation learning: A survey and a real_world case study," Sensors, vol. 21, no. 21, p. 7328, Nov. 2021.

[11] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 12, pp. 11106–11115, May 2021.

[12] Z. Hajiakhondi Meybodi, A. Mohammadi, M. Shojafar, Z. Ding, and R. Tafazolli, "TEDGE_Caching: Transformer_based edge caching towards 6G networks," arXiv preprint arXiv:2112.00633, Dec. 2021.

[13] M. Catillo, U. Villano, and M. Rak, "A survey on auto-scaling: How to exploit cloud elasticity," International Journal of Grid and Utility Computing, vol. 14, no. 1, pp. 37–50, 2023.

[14] A. Ioannou and S. Vassiliades, "A systematic survey on content caching in ICN and ICN-IoT: Challenges, approaches and strategies," Computer Networks, vol. 233, p. 109896, Sep. 2023.

[15] L. Lian, N. Chen, X. Yuan, and J. Lu, "Low_complexity collaborative caching strategy based on spatio_temporal graph convolutional model," Computer Networks, vol. 249, p. 110490, Jul. 2024.

[16] J. Zhu, R. Li, X. Chen, S. Mao, J. Wu, and Z. Zhao, "Semantics_enhanced temporal graph networks for content popularity prediction," IEEE Transactions on Mobile Computing, vol. 23, no. 8, pp. 8478–8492, Aug. 2024.