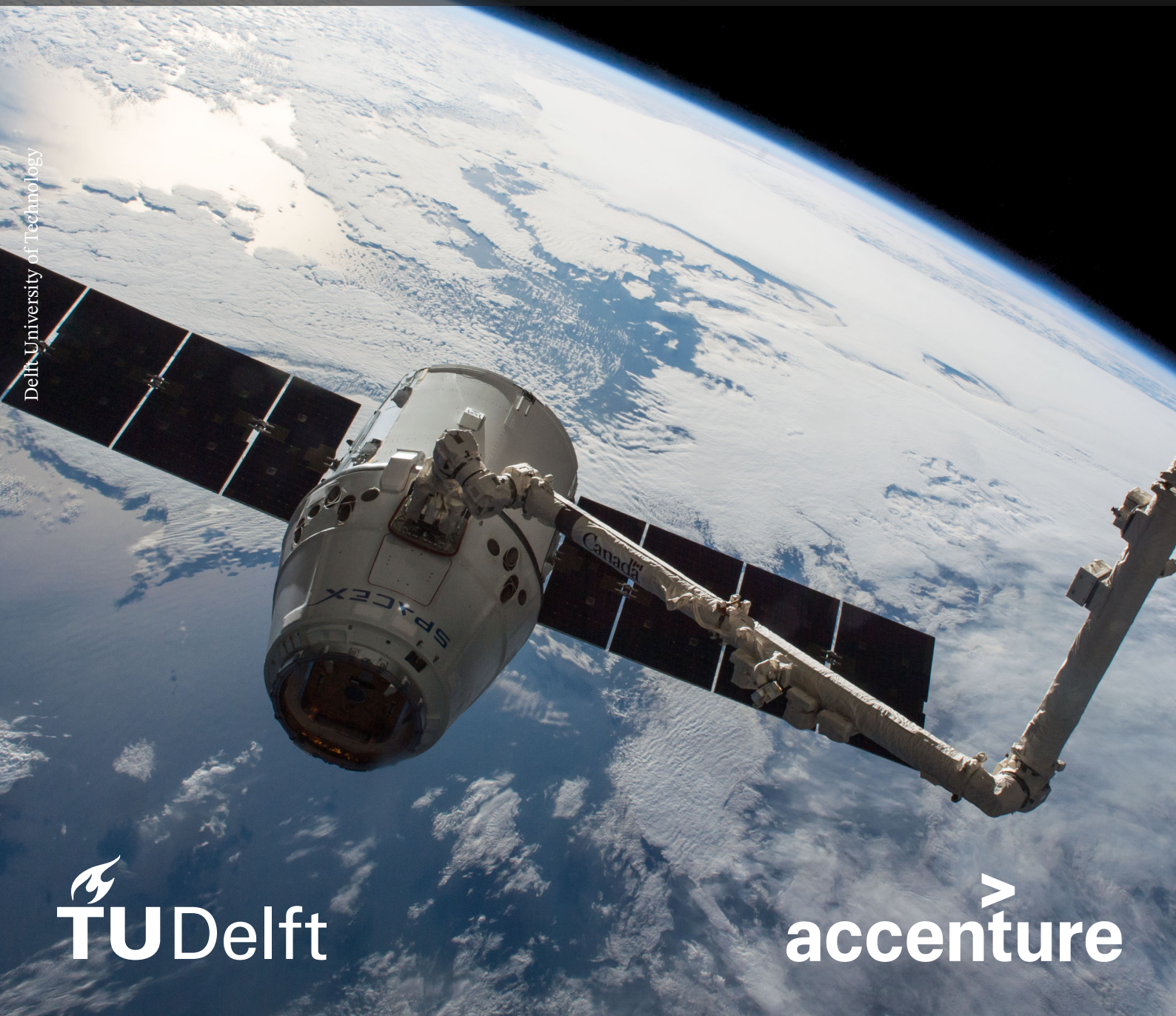
Tensor Networks for Model Predictive Control

A study of tensor train representation of the semi-explicit MPC operators

MSc Thesis

Sebastiaan Maljers



Delft University of Technology

TU Delft

accenture

Tensor Networks for Model Predictive Control

A study of tensor train representation of the semi-explicit MPC operators

by

Sebastiaan Maljers

to obtain the degree of Master of Science in Systems and Control
at the Delft University of Technology,
to be defended on *Wednesday, June 24, 2026*.

Student number:	*5045053*	
Project duration:	*February, 2026 – June, 2026*	
Faculty:	Mechanical Engineering, Delft Center for Systems and Control	
Thesis committee:	Dr. ir. *Kim Batselier*,	TU Delft, chair
	Dr. *Laura Astola*,	Accenture, supervisor
	Dr. *Koty McAllister*,	TU Delft, external member

Cover: Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft accenture

Abstract

Model predictive control (MPC) computes optimal control actions while enforcing the physical and safety limits of a system, a combination that has led to its wide adoption. Yet each action requires solving an optimisation problem within one sampling interval, a computational burden that restricts where it can be deployed. To reduce this burden, semi-explicit MPC precomputes state-independent operators. At scale, however, storage becomes the dominant cost, as the largest operator grows quadratically with the number of constraints stacked over the horizon. Because the same dynamics repeat at every step, these operators carry a regular block structure well suited to tensor train compression. This thesis presents the first tensor train formulation of semi-explicit MPC. Rewriting the semi-explicit pipeline to separate the horizon from the per-step dynamics makes it tractable to construct the operators directly in this format and to evaluate the online active-set law without ever decompressing the operators. On a benchmark that is scaled in system size and horizon, the compressed controller remains closed-loop admissible while reducing storage by up to a factor of 36. These gains come at a price. Construction in compressed format is orders of magnitude slower than its matrix counterpart. Online evaluation is 2 to 54 times slower, since every entry used by the active-set loop must be computed on demand rather than read from memory. The approach is therefore advantageous where storage, rather than computation, is the limiting resource. At long horizons the matrix-form controller no longer fits in fast memory, whereas the compressed controller, once built, does. The reported experiments stop short of that regime because, although the final controller is small, the intermediate rank growth during construction exhausts memory. Closing this gap rests on a single open problem: evaluating the inverse action that arises during operator construction while containing rank growth. Solving it would extend semi-explicit MPC to horizons beyond the reach of current methods.

Reflection on the Use of AI Tools

I completed this thesis as an intern at Accenture, where the responsible adoption of AI is encouraged as a means of boosting productivity. This aligns with my personal aspiration to produce the best work I am capable of, and these tools genuinely make that possible. They also carry real dangers, something I became especially aware of after reading the Leiden Declaration on Artificial Intelligence and Mathematics. Inspired by this, I set out below some of the strengths and weaknesses of AI in academic research, both those I faced personally and others more generally, which have guided the finalisation of this thesis.

Idea generation. Used as a sounding board, a language model is a capable sparring partner. It can stress-test an argument, offer alternative framings, and act as an auditor for complex logic, catching gaps that are easy to miss when working alone. The deepest danger also lies here. A model can produce a proof or argument that looks convincing but hides errors that are difficult to detect. Relying on reasoning one can no longer follow is where the tool stops serving the research and starts to undermine it.

Writing. AI lowers the cost of writing. It helps turn rough notes into clear prose, supports the structuring of an argument both verbally and mathematically, and, applied over short passages, improves spelling, punctuation, and readability. Because drafting becomes cheap, the author can afford far more cycles of writing, rereading, and revising than would otherwise be feasible. Left unchecked, however, it writes in an unmistakable LLM-style rhetoric, using long sentences, inflated phrasing, and a vague, instructive tone. It can coin terminology that is at best unclear and at worst incorrect. Over large spans of text it tends to drop citations or attach them carelessly, and it can introduce changes the author never intended that quietly alter the logic of an argument. It frequently builds on prior human work without acknowledging it, leading to mistaken contributions and issues of intellectual property.

Programming. Given a clear specification, a model can write code that would otherwise be out of reach, so programming ability is no longer the factor limiting which theories can be tested. It can also process and analyse large datasets, surfacing trends that are not otherwise obvious, provided the user understands the underlying method and checks the result against the data it came from. AI can also introduce workarounds that defeat the purpose of an experiment and clutter a project with superfluous code. And an author who does not continually inspect and question what is generated will gradually lose command of their own code.

A common thread runs through these dangers: AI is most valuable when it accelerates work the author can fully understand, and most hazardous when it substitutes for that understanding. Transparency and accuracy are therefore the principles that have guided its use in this thesis, in keeping with the spirit of the Leiden Declaration. AI has been used as a tool, not an author. The credit and the responsibility for this work remain mine.

Declaration

During the preparation of this thesis I made use of Claude (Opus 4), a large language model developed by Anthropic, as an assistive tool. It was used in four ways: to refine the written text for clarity, structure, and concision; to support the analysis and interpretation of the experimental data; to assist in the development and presentation of the underlying theory; and to write code from implementation structures that I specified. Every suggestion produced with this tool was critically assessed, verified against my own derivations and results, and revised where necessary. The final text has been reviewed and approved for its content, including the correctness of all definitions, derivations, results, and conclusions presented herein.

Anthropic. *Claude* (Opus 4) [large language model]. 2026. `https://claude.ai`.

Contents

List of Figures

List of Tables

List of Acronyms

ALS	alternating linear scheme
AMEn	alternating minimal energy
CPD	canonical polyadic decomposition
DMRG	density-matrix renormalization group
HJB	Hamilton–Jacobi–Bellman
KKT	Karush–Kuhn–Tucker
LTI	linear time-invariant
MALS	modified alternating linear scheme
MPC	model predictive control
OSQP	operator-splitting quadratic-program solver
PCG	preconditioned conjugate gradient
QP	quadratic program
RHS	right-hand side
SE	semi-explicit
SPD	symmetric positive definite
SVD	singular value decomposition
TT	tensor train
TT-SVD	tensor train singular value decomposition
TTM	tensor train matrix

List of Symbols

n	State dimension
m	Input dimension
N	Prediction horizon
p	Inequality constraints
Np	Inequality constraints p, stacked over the horizon N
$\boldsymbol{A}$	State-transition matrix
$\boldsymbol{B}$	Input matrix
$\boldsymbol{Q}$	State-cost weight matrix
$\boldsymbol{R}$	Input-cost weight matrix
$\boldsymbol{P}$	Terminal-cost weight matrix
$\boldsymbol{U}$	Stacked input sequence
x_0	Current state
$\boldsymbol{\lambda}$	Lagrange multipliers
$\mathcal{A}$	Active constraint set
$\boldsymbol{G}$	Condensed inequality-constraint matrix
$\boldsymbol{H}$	Condensed Hessian
$\boldsymbol{F}$	Condensed linear cost matrix
$\boldsymbol{E}$	Condensed constraint–state operator
$\boldsymbol{L}$	Linear gain operator
$\boldsymbol{M}$	Multiple-RHS operator
$\boldsymbol{S}$	Dual-coupling matrix
$\boldsymbol{T}$	Affine offset operator
$\bar{\boldsymbol{w}}$	Stacked constraint vector
d	Number of cores in a tensor train
d_t	Number of temporal cores
d_x	Number of physical cores
I_k	Row mode dimension
J_k	Column mode dimension
$R_{\max}$	Maximum rank of a tensor-train representation
$R_{\max}(\boldsymbol{S})$	Maximum rank of the tensor-train representation of the dual-coupling matrix $\boldsymbol{S}$
$R_{\max}(\boldsymbol{S})/Np$	Rank burden of the tensor-train representation of the dual-coupling matrix $\boldsymbol{S}$
δ	Operator-approximation tolerance
δ_{safe}	Largest δ at which the compressed controller remains admissible
$\delta^\star$	The smallest δ_{safe} over tested configurations
λ_E	Largest-magnitude eigenvalue of $\boldsymbol{A}$
ρ_{storage}	Compression ratio of the tensorised operator set relative to the matrix-form set
$v_{\max}$	Maximum closed-loop constraint violation

1

Introduction

Model predictive control (MPC) computes a control action by solving, at every sampling instant, a constrained optimisation problem over a finite prediction horizon. It applies only the first input of the resulting sequence, then repeats the same procedure at the next instant [1]. This formulation can deal directly with multivariable dynamics and hard constraints on states and inputs, which explains its wide use in process and motion control [2], [3]. Its main limitation is computational. At each sampling instant a quadratic program (QP) must be solved. For systems with fast sampling, high state dimension, or long prediction horizons, this online solve can determine whether the controller can be implemented in real time [4], [5].

Explicit MPC removes this online solve by precomputing the optimiser offline. The finite-horizon problem is written as a multiparametric QP, whose solution is a piecewise-affine function of the state. Online, the controller locates the active region and evaluates the corresponding affine law [6], [7]. This reduces the online work to a point-location step and a matrix-vector product, but it shifts the burden elsewhere. Constructing the solution, storing the resulting partition, and locating the active region online all scale with the number of critical regions, which grows exponentially with the number of constraints and the horizon size [7]. In practice, this confines explicit MPC to systems of modest size and short horizons, where the offline construction effort and the memory needed to store the regions remain manageable [7], [8], [9].

Semi-explicit (SE) MPC sits between these two approaches. Instead of enumerating the explicit solution over all regions, it precomputes a set of state-independent operators from the condensed QP and solves a reduced active-set problem online [4], [10], [11]. These operators are computed once and reused at every sampling instant. This avoids both the repeated full QP solve of online MPC and the combinatorial region count of explicit MPC. The formulation is well established and is used here as the reference pipeline; chapter 3 develops it in full. What matters for the argument that follows is that precomputation does not eliminate offline construction and storage costs. These costs move from a partition of the state space to the operators themselves.

For large problems, these operators dominate the controller's memory requirements. The largest is the dual-coupling matrix $\boldsymbol{S}$, of size $Np \times Np$, whose dimension grows with both the prediction horizon N and the number of inequality constraints p per step, and which accounts for most of the storage in the precomputed set. Its explicit form is given in chapter 3. Thus, the storage bottleneck that limits explicit MPC reappears as the dominant constraint of SE MPC at scale. This thesis therefore asks whether the state-independent operators admit a compact representation that reduces their storage cost while preserving the controller, and how such a representation affects the offline construction and online evaluation that rely on the same operators.

1.1. Tensor trains as a compact representation

The operators of SE MPC carry structure that comes from the way they are assembled from the system matrices over a regular prediction horizon. A representation that can exploit this structure may store them in much less memory than their matrix-form size would suggest. The tensor train (TT) format is one such representation [12]. It reshapes a large operator into a chain of smaller arrays, called cores. These cores are joined together along auxiliary indices, whose sizes are referred to as the TT ranks. When these ranks remain small, the total storage of the chain is far below that of the original operator. Standard linear-algebra operations can also be carried out directly on the cores, and a rounding step keeps the result compact at a prescribed accuracy. The technical details are developed in chapter 2; for the introduction it is enough that the format offers a controllable accuracy-versus-size trade-off for structured operators.

TTs have been used in control, but for different objects from the condensed SE operators considered here, as section 1.2 discusses. Representing those operators, and characterising when their representation reduces the storage they require, is the case taken up in this thesis; the operators themselves are defined in chapter 3.

Whether such a representation is actually beneficial depends on two properties that are not guaranteed in advance. The compressed operators must remain accurate enough that they still produce the same control action as the matrix-form ones, and the TT ranks needed to reach that accuracy must stay small enough for the format to save memory in the first place. The supporting questions below are aimed precisely at testing whether both conditions hold for the operators of SE MPC.

1.2. Related work

Tensor formats have entered control mainly through the value function of dynamic programming. Oster, Sallandt, and Schneider represent the finite-horizon value function in a hierarchical tensor format, approximating the associated Hamilton–Jacobi–Bellman (HJB) equation [13]; Dolgov, Kalise, and Kunisch solve high-dimensional HJB equations directly in the TT format [14]; and Gorodetsky, Karaman, and Marzouk perform dynamic programming in a continuous tensor decomposition to lift the curse of dimensionality in stochastic optimal control [15]. A parallel line tensorises the system model rather than the controller, either by recovering compressed nonlinear dynamics [16] or by using tensor-network methods for state estimation and identification [17]. These works establish that tensor compression can be effective in a control setting, but the tensorised object is the value function or the system model. The precomputed operators of a condensed QP, which dominate the storage of SE MPC and are the object considered in this thesis, are not addressed.

A second body of work targets the online QP solve itself. Active-set and operator-splitting solvers reduce the per-step solve time [4], [18]; condensing replaces the sparse multi-stage problem with a smaller dense one [5]; and structure-exploiting interior-point methods exploit the stage-wise banded Karush–Kuhn–Tucker (KKT) system to keep the per-iteration cost linear in the horizon [11], [19], [20]. All of these methods keep the computation online, so they never produce a precomputed operator set and therefore raise no storage question of the kind studied here. SE MPC [10] takes the opposite route. It moves the heavy linear algebra offline into a fixed operator set, and it is this precomputation that creates the storage problem at the centre of this thesis.

Explicit MPC precomputes the entire control law as a piecewise-affine map over a state-space partition [6], [7]. Because the number of regions grows combinatorially, much of the literature targets that storage problem, either through point-location structures that evaluate the map in logarithmic time [21] or through learned approximations such as deep networks that represent the piecewise-affine law with far fewer parameters [22]. Tensor compression has reached this line as well. Yáñez et al. replace the polyhedral partition by a uniform mesh, store the per-cell control laws as one large tensor, and compress it with a canonical polyadic decomposition (CPD), cutting its storage by more than 97% [23]. The pipeline remains calculate-then-compress. The full tensor must be built cell by cell before it can be compressed, and the authors name this offline construction as the main limiting factor of the approach. These techniques reduce the cost of a law that has already been enumerated. That limitation set the direction of this thesis. The direct remedy would be to construct the compressed law without ever forming the full object. For a region partition this fails for a structural reason. The partition is built by a loop over regions, each defined by its own active set, and a long sequence of small data-dependent steps gives tensor arithmetic nothing large to operate on.

Tensor formats pay off when the computation consists of a few large uniform operations. The SE operator set is built by exactly such operations, which is why this thesis takes it as the target.

Compressing large structured operators is also a mature topic in numerical linear algebra, with tensor low-rank methods surveyed by Grasedyck, Kressner, and Tobler [24] and the TT format playing a central role [12]. This literature supplies the algorithmic basis on which the present work builds. Simultaneous TT solution of many right-hand sides (RHSs) has also been studied. Coulaud, Giraud, and Iannacito carry the RHS index as an extra tensor mode and solve the enlarged system with a TT variant of GMRES [25]. This body of work, however, has not been applied to the condensed SE operators, whose construction raises the multiple-RHS inverse-action problem of chapter 4.

None of these four research lines addresses the condensed SE operator set directly. The conditions under which its TT representation is worthwhile, and the cost of constructing it in that format, therefore remain open. The research question makes this gap precise.

1.3. Research question and supporting questions

The storage, construction, and online cost of SE MPC are determined by its state-independent operators. Whether these operators admit a compact TT representation, and whether such a representation can be constructed and evaluated efficiently, determines whether tensorisation changes the scalability of the method. The thesis is organised around one research question:

> To what extent, and under what conditions, does TT representation of the condensed SE MPC operators transform the storage, offline-construction, and online-latency profile of MPC for structured large-scale linear systems?

This question separates into four supporting questions. The first identifies the accuracy needed for a compressed controller to remain usable. The remaining three questions depend on this requirement to be well posed. They then address storage, construction, and online-latency cost in turn.

1. What accuracy must the compressed operators retain for the resulting controller to remain closed-loop admissible, and what determines this requirement?
2. How does TT compression of the operators affect their storage, and what governs the ratio of compressed to matrix-form size?
3. Can the operators be constructed directly in TT form as efficiently as their compressibility suggests, and what governs the cost of doing so?
4. How does tensor-compatible online evaluation affect control latency relative to the matrix-form SE evaluator, and how does it relate to the storage achieved?

Each question is examined experimentally in chapter 5, and answered together with the main research question in chapter 6.

1.4. Contributions

This thesis develops the first pipeline for representing, constructing, and evaluating the condensed SE MPC operators in TT form, and characterises when doing so is worthwhile. It makes four contributions. The first is a tensor-compatible reformulation of the condensed SE pipeline. Every stage, from the stacked primitives to the final operator set, is rewritten in a form that separates the horizon from the per-step physics, with the temporal-shift Kronecker-sum form of equation (4.6) as the main tool. This separation is the structure the TT format compresses, and it is what lets representation, construction, and evaluation all stay within the format. The second is a shape-plan formalism that fixes how each operator's horizon and physical dimensions are arranged into tensor cores. The third is a tensor-compatible active-set evaluator that runs the online control law entirely on the compressed cores. The fourth is a set of structural predictions, derived in section 4.10, that link the accuracy, storage, construction cost, and online cost to the rank and conditioning of the operators, and that the experiments then test. Supporting these, a multiple-RHS solver assembled from standard components (section 4.7) builds the operators directly in TT form.

The verdict is conditional. The representation improves one of the three costs and not the other two. Compression at admissible accuracy reaches 36×, so storage falls. Construction does not get cheaper, because the rounding that controls rank growth during TT arithmetic costs more than the dense linear algebra it replaces, and online evaluation does not overtake the matrix-form controller within the tested range. The method is therefore worthwhile in one regime, the storage-bound long horizon, where the matrix-form operators no longer fit in fast memory while the compressed controller does.

The scope is narrow. Every quantitative claim depends on the TT ranks of the specific operators, which in turn depend on the shape plan and the system structure, so the findings characterise when TT representation is worthwhile for the condensed SE operators of a structured linear system rather than for MPC in general. Section 6.6 states the limitations in full and discusses the choice of baselines.

1.5. Outline

The remainder of this thesis is organised as follows. Chapter 2 introduces the TT format and the operations used in later chapters. Chapter 3 derives the condensed SE MPC formulation and the state-independent operators that the remainder of the thesis takes as its object of study. Chapter 4 develops the tensorised pipeline: the points at which compression can be introduced, the shape plan, the multiple-RHS solver used to construct the operators, and the tensor-compatible online evaluator, while identifying the structural bottleneck each route faces. Chapter 5 reports the experimental study, a dependency-ordered sequence of experiments that test the four supporting questions. Chapter 6 answers the supporting questions and the main research question, discusses the findings, and sets out directions for future work.

2

Tensor Network Preliminaries

A TT represents a tensor as a chain of small three-index objects called cores, with one core assigned to each dimension of the tensor. The size of this chain is controlled by a set of integers, the TT ranks, which describe how strongly neighbouring dimensions are coupled. When these ranks remain small, the tensor can be stored and manipulated far more cheaply than its uncompressed size would suggest. This chapter introduces the format, the operations defined on it, and the structural property that allows its ranks to stay small.

2.1. From matrices to tensors

A tensor is a multidimensional array of order three or higher. It extends the objects of ordinary linear algebra. Scalars, vectors, and matrices are arrays of order zero, one, and two, indexed by zero, one, and two indices respectively. A matrix arranges numbers in a grid addressed by a row index and a column index. An order-d tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ arranges them in a d-dimensional grid addressed by d indices (figure 2.1). Each of the d directions is a *mode*. The index $i_k \in \{1, \ldots, I_k\}$ runs along mode k, and the integer I_k is the *mode dimension* of that mode. The typographic conventions used throughout the thesis are collected in table 2.1 [26], [27].

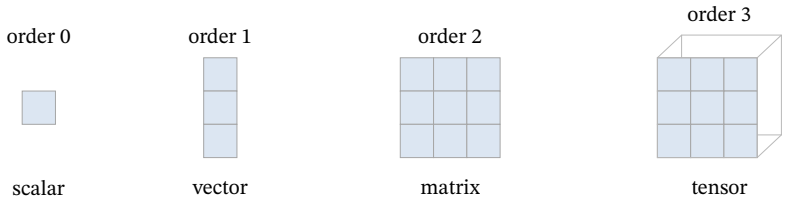


Figure 2.1: Scalars, vectors, matrices, and tensors as arrays of increasing order.

Table 2.1: Typographic conventions used throughout the thesis.

Object	Symbol	Convention
Scalar	x	lowercase italic
Vector	$\boldsymbol{x}$	lowercase bold
Matrix	$\boldsymbol{X}$	uppercase bold
Tensor (order ≥ 3)	$\mathcal{X}$	calligraphic
TT core k	$\mathcal{G}^{(k)}$	calligraphic with superscript
Mode index	i_k	index running along mode k
Mode dimension	I_k	size of mode k
TT rank	R_k	internal dimension between cores k and $k{+}1$
Approximation	$\widetilde{\boldsymbol{X}}, \widetilde{\mathcal{X}}$	tilde marks an approximation

2.1.1. Tensor Network Diagrams

Tensor expressions are often easier to read as diagrams than as indexed sums, so the remainder of this thesis uses the diagrammatic notation shown in figure 2.2. A node denotes a tensor, each edge incident to it denotes one of its indices, and an edge joining two nodes denotes summation over the shared index, an operation called a *contraction* [26], [27]. A free edge is an index of the result, so the number of free edges equals the order of the object represented by the diagram. The right panel of figure 2.2 illustrates this on the familiar matrix product $\boldsymbol{AB}$. Two nodes are joined by a contracted edge for the summation index, and the two free edges carry the row and column indices of the result.

Figure 2.2: Diagrammatic notation. A node is a tensor, an edge is an index, and a shared edge is a contraction.

2.1.2. Three useful operations

Three operations from multilinear algebra are used throughout the rest of the thesis: *vectorisation*, the *mode-k unfolding*, and the *Kronecker product*. The first two are different ways of flattening a tensor into a vector or a matrix. The third builds a structured matrix from two smaller ones and underlies the TT matrix format of section 2.4.

Definition 2.1 (Vectorisation)**.** The vectorisation $\mathrm{vec}(\mathcal{X}) \in \mathbb{R}^{I_1 \cdots I_d}$ of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ stacks its entries into a single column vector in a fixed index order. For a matrix this reduces to stacking its columns (figure 2.3) [26].

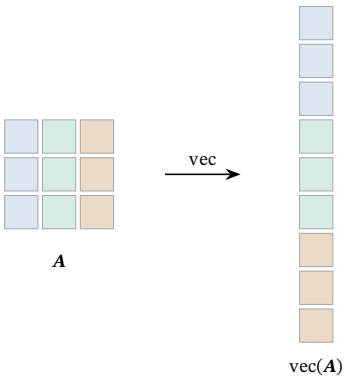


Figure 2.3: Vectorisation of a matrix by stacking its columns.

The mode-k unfolding flattens a tensor into a matrix along a chosen mode. It is the operation on which the TT decomposition in section 2.3 is built.

Definition 2.2 (Mode-k unfolding)**.** The mode-k unfolding $\boldsymbol{X}_{(k)} \in \mathbb{R}^{I_k \times (I_1 \cdots I_{k-1} I_{k+1} \cdots I_d)}$ of a tensor $\mathcal{X}$ is the matrix whose columns are the mode-k fibres of $\mathcal{X}$, the vectors obtained by fixing every index except i_k (figure 2.4) [26].

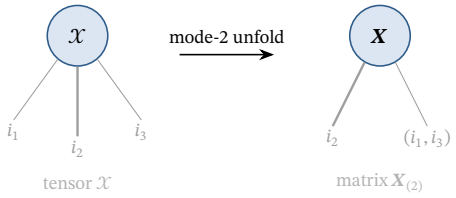


Figure 2.4: Mode-2 unfolding of a third-order tensor. The selected mode becomes the row index of the matrix; the remaining modes are grouped into the column index.

The Kronecker product takes two matrices and produces a single larger one, and will appear repeatedly, both in the matrix TT format of section 2.4 and in the mode-separable operators that motivate the whole construction (section 2.7).

Definition 2.3 (Kronecker product)**.** For $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p\times q}$, the Kronecker product $\boldsymbol{A}\otimes\boldsymbol{B} \in \mathbb{R}^{mp\times nq}$ is the block matrix whose (i, j) block is $a_{ij}\boldsymbol{B}$ [26].

Splitting the row and column indices into the pairs (i_A, i_B) and (j_A, j_B) shows the Kronecker product as the entrywise factorisation $(\boldsymbol{A} \otimes \boldsymbol{B})_{(i_A i_B),(j_A j_B)} = a_{i_A j_A}\, b_{i_B j_B}$. In diagrams it is the outer product of $\boldsymbol{A}$ and $\boldsymbol{B}$, that is, two nodes joined by a single edge of rank one (figure 2.5). This is the simplest example of the chained decompositions that the rest of the chapter is built on.

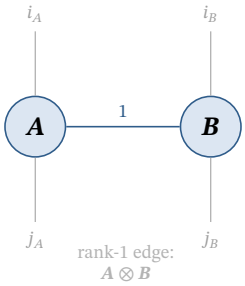


Figure 2.5: The Kronecker product as the outer product of $\boldsymbol{A}$ and $\boldsymbol{B}$.

2.2. Tensor decompositions

A tensor decomposition writes a high-order tensor as a network of smaller component tensors. Three decompositions are standard [26], [27]. The CPD writes a tensor as a sum of R rank-one terms; the Tucker decomposition attaches a small matrix to each mode of a smaller core tensor; and the TT decomposition, used throughout the rest of this chapter, arranges its cores in a linear chain (figure 2.6). Their main differences are summarised in table 2.2.

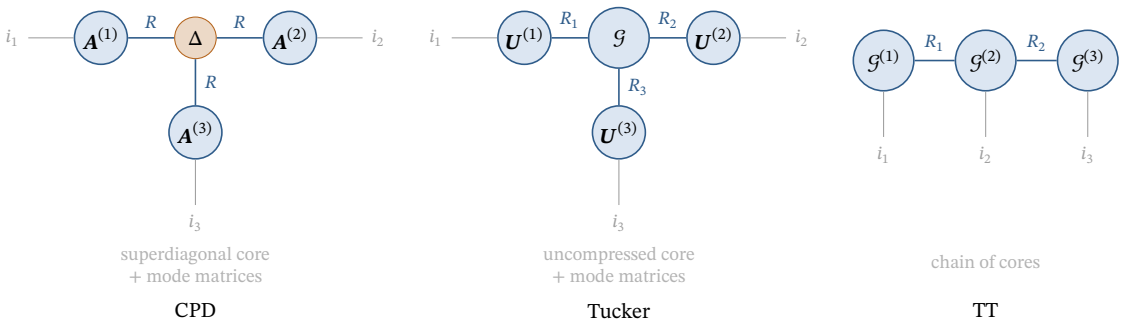


Figure 2.6: The three decompositions of an order-three tensor.

Table 2.2: The three decompositions for an order-d tensor with mode dimension n and rank r.

	CPD	Tucker	TT
Storage	dnr	$dnr + r^d$	dnr^2
Unique, interpretable components	yes	no	no
Best low-rank approximation exists	no	yes	yes
Growth with the order d	linear	core grows as r^d	linear
Arithmetic without the uncompressed tensor	limited	limited	native, with rounding

These decompositions make different trade-offs. The CPD has unique, interpretable components, determined by the tensor itself under mild conditions [26], which makes it useful when these components carry physical meaning. However, the set of tensors of rank at most R is not closed, so a best rank-R approximation may fail to exist [28]. Tucker restores a well-posed best approximation through the higher-order singular value decomposition (SVD) [27]. Neither format, however, is as convenient for computation as the TT. Adding, multiplying, and applying operators in CPD or Tucker form is awkward, whereas the chain structure of the TT performs these operations on the cores and returns the result to compact form by rounding [12]. Because its storage also stays linear in the order d, the TT is used throughout the remainder of the thesis.

2.3. The tensor train format

The TT represents an order-d tensor by a chain of order-three cores, one for each mode.

Definition 2.4 (Tensor-train format)**.** A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is in TT format if its entries are

$$\mathcal{X}(i_1, \ldots, i_d) = \mathcal{G}^{(1)}(i_1)\, \mathcal{G}^{(2)}(i_2) \cdots \mathcal{G}^{(d)}(i_d), \tag{2.1}$$

where each *core* $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ is an order-three tensor and $\mathcal{G}^{(k)}(i_k) \in \mathbb{R}^{R_{k-1} \times R_k}$ is the matrix obtained by fixing its middle index. The boundary ranks are $R_0 = R_d = 1$, so the product in equation (2.1) is a scalar. The integers R_k are the *TT ranks*, an external (physical) edge i_k is an index of the tensor, and an internal edge is summed between neighbouring cores (figure 2.7).

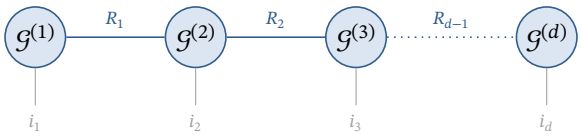


Figure 2.7: A TT as a chain of order-three cores.

A TT stores $\sum_k R_{k-1} I_k R_k$ parameters, or at most dIR^2 when $I = \max_k I_k$ is the largest mode dimension and $R = \max_k R_k$ is the largest rank. This is linear in the order d, compared with the $\prod_k I_k = I^d$ entries of the uncompressed tensor. Thus, the format removes the exponential dependence on d whenever the ranks remain bounded.

Any tensor can be placed in this format, and truncated to a chosen accuracy, by the tensor train singular value decomposition (TT-SVD) (Algorithm 1). The algorithm sweeps along the cores, taking one SVD for each of the $d - 1$ internal TT ranks and retaining the singular values allowed by a predefined tolerance.

Algorithm 1 TT-SVD [12].

Require: tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, tolerance ε
Ensure: cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$
1: $C \leftarrow \mathcal{X}, \quad R_0 \leftarrow 1$
2: **for** $k = 1$ to $d-1$ **do**
3: reshape C into the matrix $C \in \mathbb{R}^{R_{k-1} I_k \times (I_{k+1} \cdots I_d)}$
4: $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\top] \leftarrow \mathrm{SVD}(C)$, truncated to rank R_k at tolerance $\varepsilon/\sqrt{d-1}$
5: $\mathcal{G}^{(k)} \leftarrow \mathrm{reshape}(\boldsymbol{U}, R_{k-1} \times I_k \times R_k)$
6: $C \leftarrow \boldsymbol{\Sigma}\boldsymbol{V}^\top$
7: **end for**
8: $\mathcal{G}^{(d)} \leftarrow \mathrm{reshape}(C, R_{d-1} \times I_d \times 1)$

The truncation error is bounded. If the singular values discarded at each TT rank contribute at most $\varepsilon/\sqrt{d-1}$ in Frobenius norm, the resulting TT $\widetilde{\mathcal{X}}$ satisfies $\|\mathcal{X} - \widetilde{\mathcal{X}}\|_F \le \varepsilon \|\mathcal{X}\|_F$, and its error exceeds the best possible TT of the same ranks by at most a factor of $\sqrt{d-1}$ [12]. The TT ranks are not chosen in advance. Each is set based on the number of singular values an unfolding needs to reach the tolerance. A tensor compresses well when those singular values decay quickly. Compressibility in the TT format is therefore a property of the unfoldings, and chapter 4 builds on it.

For $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ the k-th rank returned by the TT-SVD (Algorithm 1) equals the rank of the k-th unfolding $\boldsymbol{X}_{(1:k)} \in \mathbb{R}^{(I_1 \cdots I_k) \times (I_{k+1} \cdots I_d)}$, the matrix that groups the first k modes as rows, and is therefore bounded by the smaller of its two sides,

$$R_k \;\le\; \min\Big(\prod_{l \le k} I_l,\; \prod_{l > k} I_l\Big), \qquad R_{\max} \;\le\; \max_{1 \le k < d} \min\Big(\prod_{l \le k} I_l,\; \prod_{l > k} I_l\Big). \tag{2.2}$$

This is the standard unfolding characterisation of the TT ranks [12]. It is recalled in this form because the shape plans of section 4.2 turn it into a design objective, using the bound to score candidate factorisations.

2.4. The tensor train matrix format

A matrix in this format carries a row index and a column index for each mode. Turning an ordinary matrix into that form is called *tensorisation*. The single row index and the single column index are each broken into d sub-indices, and the resulting row and column groups are paired mode by mode.

Definition 2.5 (Matrix tensorisation)**.** Let the row and column dimensions of $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ admit factorisations $I = I_1 \cdots I_d$ and $J = J_1 \cdots J_d$. Tensorising $\boldsymbol{A}$ means re-addressing its entries by replacing the row index $i \in \{1, \ldots, I\}$ with the multi-index $(i_1, \ldots, i_d)$, with $i_k \in \{1, \ldots, I_k\}$, and similarly the column index j with $(j_1, \ldots, j_d)$, and then pairing the k-th row and column groups into a single mode of size (I_k, J_k). The result is an order-d array with paired modes, addressed as $\boldsymbol{A}\big((i_1, \ldots, i_d), (j_1, \ldots, j_d)\big)$ [12].

The same matrix admits many tensorisations, depending on how many sub-indices are used and how they are ordered, and each tensorisation exposes different unfoldings. Because the ranks of Definition 2.4 are read from those unfoldings, tensorisation is not just notation. It determines the ranks seen by TT compression [12], [27]. This choice is the freedom that section 4.2 turns into a design variable for storage. Pairing the modes of a tensorisation gives the tensor train matrix (TTM) format.

Definition 2.6 (Tensor-train-matrix format)**.** A matrix $\boldsymbol{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (J_1 \cdots J_d)}$ is in TTM format if

$$\boldsymbol{A}\big((i_1, \ldots, i_d), (j_1, \ldots, j_d)\big) = \mathcal{G}^{(1)}(i_1, j_1) \cdots \mathcal{G}^{(d)}(i_d, j_d), \tag{2.3}$$

with cores $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$ each carrying a row edge i_k and a column edge j_k (figure 2.8).

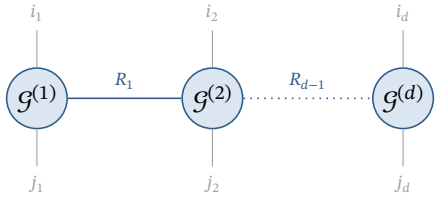


Figure 2.8: A TTM: each core carries a row edge and a column edge.

The storage of the format follows directly from the core sizes.

A TT vector with cores $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1}\times I_k\times R_k}$ stores $\sum_{k=1}^{d} R_{k-1}I_kR_k$ parameters, and a TTM with cores $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1}\times I_k\times J_k\times R_k}$ stores

$$B_{\mathrm{TTM}} = \sum_{k=1}^{d} R_{k-1}\, I_k\, J_k\, R_k \tag{2.4}$$

parameters. With $R = \max_k R_k$ and $\bar{m} = \max_k I_kJ_k$ this is at most $d\,R^2\,\bar{m}$, linear in the order d against the $\prod_k I_kJ_k$ entries of the uncompressed matrix. The count depends on the mode dimensions I_k, J_k and the TT ranks R_k. For a structured operator, the ranks remain small while the mode dimensions carry the size, so the storage can be far below the uncompressed count.

A single Kronecker product corresponds to TT rank one in equation (2.3), and a sum of k Kronecker products has TTM rank at most k [12]. The TT rank of a matrix therefore measures how far it is from a single Kronecker product, a perspective made concrete by the example below.

Example 2.1 (Tensor-train rank and storage of a structured operator)**.** Consider the 8×8 second-difference matrix

$$\boldsymbol{L} = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}. \tag{2.5}$$

To place $\boldsymbol{L}$ in the TTM format, split both dimensions as $8 = 2\cdot 2\cdot 2$, so that each row index and each column index splits into three sub-indices. Under this tensorisation $\boldsymbol{L}$ can be written as a sum of three Kronecker products of 2×2 matrices, which means that its TTM rank is three and that the decomposition is exact. The three cores then have shapes $(1,2,2,3)$, $(3,2,2,3)$, and $(3,2,2,1)$. The first and last cores each carry one boundary rank of size one and one internal rank of size three, while the middle core has two internal ranks of size three (figure 2.9).

Applying equation (2.4) to these shapes gives a storage of $12+36+12 = 60$ parameters against the 64 entries of the uncompressed matrix. At this size the saving is marginal, because the rank-three cores nearly fill the operator. The benefit appears as the operator grows. The same second-difference structure on $N = 2^d$ grid points keeps the TTM rank at three, so the storage is $36d-48$ against $N^2 = 4^d$ entries in the uncompressed matrix (table 2.3). The compression ratio therefore grows indefinitely as the operator scales, because the rank stays fixed while N^2 grows. chapter 5 measures the same effect for the SE operators.

Table 2.3: TTM storage of the second-difference operator at fixed rank three, against the uncompressed matrix, as the size $N = 2^d$ grows.

Size N	Uncompressed N^2	B_{TTM}	Ratio
8	64	60	1.1×
16	256	96	2.7×
32	1024	132	7.8×
64	4096	168	24.4×
128	16384	204	80.3×

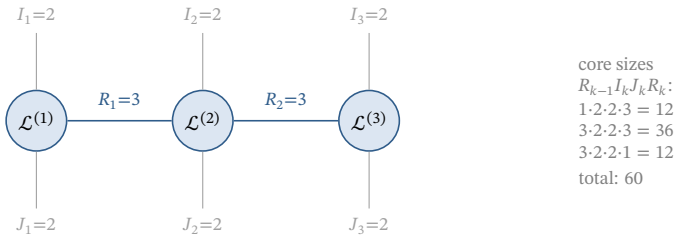


Figure 2.9: TTM representation of the 8×8 second-difference matrix $\boldsymbol{L}$ under the split $(2,2,2) \times (2,2,2)$. $\boldsymbol{L}$ is split into three cores, with row and column indices of size two and ranks of three. The boundary ranks $R_0 = R_3 = 1$ are omitted from the diagram because an edge of size one is a trivial dimension that contributes nothing to the contraction; it is still counted in the per-core storage on the right, giving a total of $12 + 36 + 12 = 60$ parameters against the 64 entries of the matrix.

2.5. Arithmetic in tensor train format

The operations of linear algebra carry over to TTs. These operations are performed directly on the cores, without forming the uncompressed tensor. The subsections below treat the sum, the products, the inner product and norm, and the rounding operation that controls the rank growth produced by these operations.

2.5.1. Computing on the cores

A sum or product of TTs is assembled core by core, so an operator too large to store in uncompressed form can still be added or applied to another tensor [12]. The result is exact, but its TT ranks are typically larger than necessary. A rounding step (section 2.5.5) restores a compact representation. Whether the format is cheaper than the uncompressed alternative depends on how much the ranks grow before rounding and on the cost of the rounding.

2.5.2. Sum

The sum of two TTs is formed by placing the operands' cores block-diagonally, mode by mode (figure 2.10). The TT ranks of the result are the sums of the operands' ranks, $R_k = R_k^{\mathrm{A}} + R_k^{\mathrm{B}}$, so addition is exact and rank-additive. A sum of many TTs therefore inflates the ranks linearly in the number of terms, and is followed by rounding.

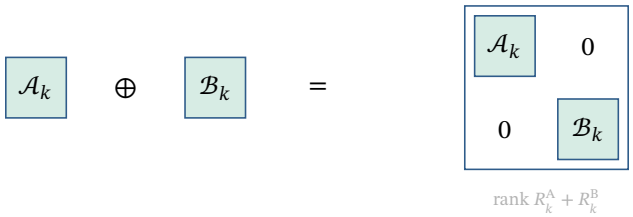


Figure 2.10: Sum of two TTs by block-diagonal stacking of cores.

2.5.3. Matrix-vector and matrix-matrix products

Applying a TTM to a TT vector contracts their cores mode by mode. Multiplying two TTMs contracts the column index of one with the row index of the other, mode by mode (figure 2.11). The result is again a TT, with ranks equal to the products of the operands' ranks, $R_k = R_k^{\mathrm{A}} R_k^{\mathrm{B}}$. Products are therefore rank-multiplicative. They grow the ranks faster than sums, so rounding after a product is especially important.

2.5.4. Inner product, norm, and projection

The inner product $\langle \mathcal{X}, \mathcal{Y} \rangle$ of two TTs is computed by contracting their two chains together, one core at a time, without forming either uncompressed tensor. The Frobenius norm follows as $\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}$. When the cores are orthogonalised, every core but one is orthonormal, so the norm reduces to the Frobenius norm of that single core. Rounding orthogonalises the chain in any case, so the approximation error of a rounded

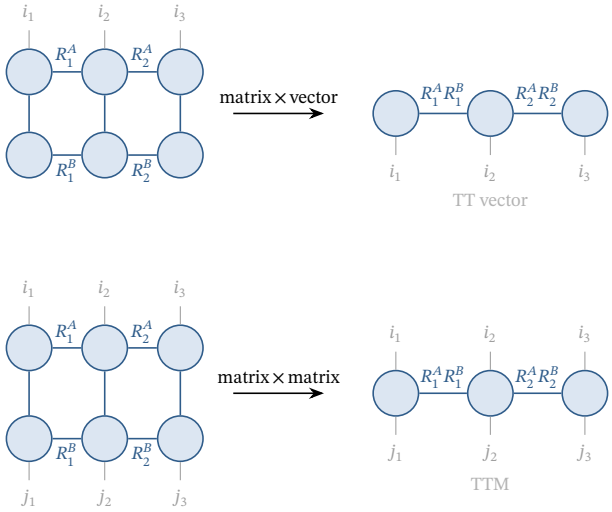


Figure 2.11: Products in the TT format, formed by contracting cores mode by mode. The ranks of the result are the products of the operands' ranks, $R_k = R_k^A R_k^B$. Top: a TTM applied to a TT vector, leaving one free edge per output core. Bottom: a product of two TTMs, leaving a row and a column edge per output core.

operator can be read from one core rather than recomputed from the full tensor. This provides the cheap error control used in the constructions of chapter 4. The orthogonal projection of a tensor onto a TT subspace is assembled from the same contractions.

2.5.5. Rounding

Rounding recompresses a TT whose ranks have grown, producing a smaller TT with prescribed accuracy. It is the TT-SVD applied to a tensor that is already in the TT format. After orthogonalising the cores, one sweep of truncated SVDs reduces each rank to the value set by the tolerance (figure 2.12).

Definition 2.7 (Tensor-train rounding)**.** Given a TT $\widetilde{\mathcal{X}}$ and a tolerance ε, rounding returns a TT $\mathcal{X}_\varepsilon$ of minimal ranks with $\|\widetilde{\mathcal{X}} - \mathcal{X}_\varepsilon\|_F \leq \varepsilon \|\widetilde{\mathcal{X}}\|_F$, computed by orthogonalisation followed by a sweep of truncated SVDs [12].

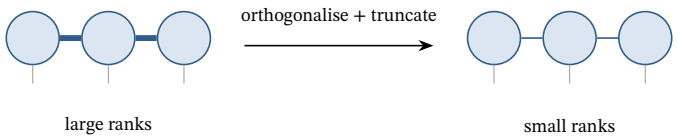


Figure 2.12: Rounding: orthogonalisation followed by a truncation sweep that thins each rank to the value set by the tolerance.

The cost of rounding is $\mathcal{O}(d\, I\, R^3)$ in the order d, the maximum mode dimension I, and the maximum TT rank R [12]. It is therefore cheap when the ranks are small and expensive when they are large.

2.6. Tensor train linear systems

Several constructions in chapter 4 require solving a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ in which the operator $\boldsymbol{A}$ is a TTM, the RHS $\boldsymbol{b}$ is a TT vector, and the solution $\boldsymbol{x}$ is sought in the same format. Forming $\boldsymbol{A}$ in uncompressed form is exactly what the format is meant to avoid, so the system is solved without leaving the tensor representation.

The standard approach fixes all cores of $\boldsymbol{x}$ except one and solves for the remaining core. Holding the other cores fixed projects the global system onto a small local system in the entries of that single core, which is solved exactly. Sweeping back and forth along the cores updates each core in turn. The projected local operator is symmetric positive definite (SPD) whenever $\boldsymbol{A}$ is, so each local solve is a small SPD system. The methods differ in how much rank adaptivity they allow. The alternating linear scheme (ALS) fixes the ranks in advance [29]; its two-site variant (modified alternating linear scheme (MALS), equivalently density-

matrix renormalization group (DMRG)) merges neighbouring cores so the ranks adapt during the sweep [29]; and the alternating minimal energy (AMEn) augments each core with a residual direction to combine rank adaptivity with one-site cost [30]. All three solve a single RHS. Chapter 4 extends the local-subproblem idea to the multiple-RHS inverse-action problem required by the SE operators.

2.7. Tensor train compatibility

Three properties established above make the TT useful as a working representation. First, its arithmetic is closed. Sums, products, inner products, and projections of TTs are again TTs, and rounding returns them to compact form [12]. Second, its truncation is controlled. The TT-SVD gives a decomposition at any prescribed accuracy with a known error bound. Third, its storage, equation (2.4), is fixed by the mode dimensions and the TT ranks, so it follows directly from the structure of the operator.

The format does not help in every case [24], [31]. It offers no saving when the operator's unfoldings have high rank, since the TT ranks inherit that rank and the storage of equation (2.4) matches or exceeds the uncompressed count; when the modes are not separable, so that no factorisation can uncover low-rank structure; when a computation's intermediate ranks grow prohibitively, so that working memory is set by the path rather than the endpoint; or when the rounding step needed to restore low rank after an operation costs more than the operation itself.

The structure possessed by the SE operators is mode separability of the kind produced by Kronecker sums. An operator of the form $\boldsymbol{A}_1 \otimes \boldsymbol{I} \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{A}_2 \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{A}_3$ acts on one mode at a time, and such operators have small, bounded TT ranks, as the second-difference operator of Example 2.1 already showed. Chapter 3 writes the operators $\boldsymbol{L}$, $\boldsymbol{M}$, $\boldsymbol{S}$, and $\boldsymbol{T}$ in this form, which brings them within reach of the tensorised pipeline of chapter 4.

Table 2.4 maps each tool introduced in this chapter to its role in the tensorised pipeline of chapter 4.

Table 2.4: Each tool from this chapter and its role later in the thesis.

Concept	**Role later in the thesis**
Tensorisation (Definition 2.5)	Re-indexes the MPC operators for the TT/TTM format under a shape plan (section 4.2).
TT and TTM format (Definitions 2.4 and 2.6)	Store the operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$ compactly (section 5.7).
TT-SVD and rounding (Algorithm 1 and Definition 2.7)	Compress the operators and control rank growth in every construction route (section 4.3).
TTM product (section 2.5.3)	Forms all operator products during construction, such as $\boldsymbol{GM}$ and $\boldsymbol{GL}$ during condensation (section 4.5.2).
Extraction and slicing	Obtain the active block $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}$ and selected columns online (section 4.8).
Tensor linear systems (section 2.6)	Solve the inverse-action systems $\boldsymbol{HL} = \boldsymbol{F}$ and $\boldsymbol{HM} = \boldsymbol{G}^\top$ (section 4.6).

3

Semi-Explicit MPC: The Reference Pipeline

SE MPC precomputes a set of state-independent operators offline and evaluates a small active-set problem online. This chapter recalls that construction for the condensed QP [4], [6], [10] and isolates the operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}, \bar{\boldsymbol{w}}$ that chapter 4 represents as TTs. The one modification introduced here is the global-operator packaging of section 3.5, which is what allows each operator to be treated as a single tensor object in the next chapter.

3.1. Linear MPC and the receding horizon policy

Consider the discrete-time linear system

$$x_{k+1} = \boldsymbol{A}x_k + \boldsymbol{B}u_k, \qquad x_k \in \mathbb{R}^n,\ u_k \in \mathbb{R}^m, \tag{3.1}$$

with state-cost weight $\boldsymbol{Q} \succeq 0$ and input-cost weight $\boldsymbol{R} \succ 0$. Over a horizon of N steps from the current state x_0, MPC solves the optimal control problem

$$\min_{u_0,\dots,u_{N-1}} \ \sum_{k=1}^{N-1} x_k^\top \boldsymbol{Q} x_k + x_N^\top \boldsymbol{P} x_N + \sum_{k=0}^{N-1} u_k^\top \boldsymbol{R} u_k \quad \text{s.t.} \quad \boldsymbol{M}_x x_k + \boldsymbol{M}_u u_k \le \boldsymbol{w}, \tag{3.2}$$

subject to the dynamics equation (3.1). The terminal state x_N is weighted by the solution $\boldsymbol{P}$ of the discrete algebraic Riccati equation, which supplies the infinite-horizon tail cost and is standard for closed-loop stability [32]. The stage constraints $\boldsymbol{M}_x x_k + \boldsymbol{M}_u u_k \le \boldsymbol{w}$ in equation (3.2) collect the box bounds on states and inputs into p inequalities per step. In receding-horizon operation the problem is solved at each instant, only the first input $u_0^\star$ is applied, and the procedure repeats at the next state [32], [33].

3.2. The condensed pipeline

Writing each predicted state x_k as a linear function of x_0 and the inputs $u_0, \dots, u_{k-1}$ and substituting into equation (3.2) turns the problem into a QP in the inputs alone.

3.2.1. Stacked prediction

Stacking the inputs as $\boldsymbol{U} = [u_0^\top, \dots, u_{N-1}^\top]^\top \in \mathbb{R}^{Nm}$ and the predicted states as $\bar{\boldsymbol{X}} = [x_1^\top, \dots, x_N^\top]^\top$, the dynamics equation (3.1) give the stacked prediction

$$\bar{\boldsymbol{X}} = \bar{\boldsymbol{A}} x_0 + \bar{\boldsymbol{B}} \boldsymbol{U}, \tag{3.3}$$

with the stacked prediction matrices

$$\bar{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix}, \qquad \bar{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{B} & 0 & \cdots & 0 \\ \boldsymbol{AB} & \boldsymbol{B} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}^{N-2}\boldsymbol{B} & \cdots & \boldsymbol{B} \end{bmatrix}. \tag{3.4}$$

The cost weights stack into the block-diagonal $\bar{\boldsymbol{Q}} = \operatorname{diag}(\boldsymbol{Q}, \dots, \boldsymbol{Q}, \boldsymbol{P})$ and $\bar{\boldsymbol{R}} = \boldsymbol{I}_N \otimes \boldsymbol{R}$, and the stage constraints to $\bar{\boldsymbol{M}}_x = \boldsymbol{I}_N \otimes \boldsymbol{M}_x$, $\bar{\boldsymbol{M}}_u = \boldsymbol{I}_N \otimes \boldsymbol{M}_u$, and $\bar{\boldsymbol{w}} = [\boldsymbol{w}^\top, \dots, \boldsymbol{w}^\top]^\top$, where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix.

3.2.2. Condensed operators

The stacked prediction equation (3.3) replaces the predicted states by a linear function of $\boldsymbol{U}$ and x_0, so substituting it into the cost and into the stage constraints of equation (3.2) eliminates the states entirely. The remaining problem depends only on the input sequence $\boldsymbol{U}$ and on the current state x_0, which is what makes it a single QP parametrised by the state [6], [10].

The cost in equation (3.2) can be grouped into a stacked stage cost $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{X}}$ and an input cost $\boldsymbol{U}^\top \bar{\boldsymbol{R}} \boldsymbol{U}$, with the stacked weight matrices defined in section 3.2.1. Replacing $\bar{\boldsymbol{X}}$ by $\bar{\boldsymbol{A}} x_0 + \bar{\boldsymbol{B}} \boldsymbol{U}$ and discarding the term that depends on x_0 alone, which is constant in the minimisation over $\boldsymbol{U}$, gives

$$J(\boldsymbol{U}, x_0) = \boldsymbol{U}^\top \big(\bar{\boldsymbol{B}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{B}} + \bar{\boldsymbol{R}}\big) \boldsymbol{U} + 2\, x_0^\top \bar{\boldsymbol{A}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{B}}\, \boldsymbol{U}. \tag{3.5}$$

Matching equation (3.5) term by term with the standard quadratic form $\frac{1}{2}\boldsymbol{U}^\top \boldsymbol{H} \boldsymbol{U} + x_0^\top \boldsymbol{F}^\top \boldsymbol{U}$ identifies the condensed Hessian and the linear cost matrix,

$$\boldsymbol{H} = 2\big(\bar{\boldsymbol{B}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{B}} + \bar{\boldsymbol{R}}\big), \qquad \boldsymbol{F} = 2\, \bar{\boldsymbol{B}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{A}}. \tag{3.6}$$

The Hessian inherits positive definiteness from $\bar{\boldsymbol{R}} \succ 0$, so $\boldsymbol{H} \succ 0$ regardless of the choice of $\bar{\boldsymbol{Q}} \succeq 0$ or of the system matrices. The rest of the pipeline relies on this property repeatedly.

The stage constraints stack into $\bar{\boldsymbol{M}}_x \bar{\boldsymbol{X}} + \bar{\boldsymbol{M}}_u \boldsymbol{U} \leq \bar{\boldsymbol{w}}$. Substituting the stacked prediction a second time and rearranging gives $\big(\bar{\boldsymbol{M}}_x \bar{\boldsymbol{B}} + \bar{\boldsymbol{M}}_u\big)\boldsymbol{U} \leq \bar{\boldsymbol{w}} - \bar{\boldsymbol{M}}_x \bar{\boldsymbol{A}} x_0$, which has the form $\boldsymbol{G}\boldsymbol{U} \leq \bar{\boldsymbol{w}} + \boldsymbol{E} x_0$ with

$$\boldsymbol{G} = \bar{\boldsymbol{M}}_x \bar{\boldsymbol{B}} + \bar{\boldsymbol{M}}_u, \qquad \boldsymbol{E} = -\bar{\boldsymbol{M}}_x \bar{\boldsymbol{A}}. \tag{3.7}$$

The constraint RHS $\bar{\boldsymbol{w}}$ is the stacked stage bound carried through unchanged, while the state dependence of the constraints is collected in the single matrix $\boldsymbol{E}$.

3.2.3. The condensed quadratic program

Collecting equations (3.6) and (3.7) expresses the optimal control problem as the parametric QP

$$\boldsymbol{U}^\star(x_0) = \arg\min_{\boldsymbol{U}} \ \tfrac{1}{2}\boldsymbol{U}^\top \boldsymbol{H} \boldsymbol{U} + x_0^\top \boldsymbol{F}^\top \boldsymbol{U} \quad \text{s.t.} \quad \boldsymbol{G}\boldsymbol{U} \leq \bar{\boldsymbol{w}} + \boldsymbol{E} x_0. \tag{3.8}$$

The Hessian $\boldsymbol{H}$, the constraint matrix $\boldsymbol{G}$, and the offset $\bar{\boldsymbol{w}}$ are fixed by the system, cost, and constraint data. The state x_0 enters only through the linear cost term and through the constraint RHS. The receding-horizon law applies the first m entries of $\boldsymbol{U}^\star(x_0)$ at each sampling instant.

3.3. KKT decomposition and the semi-explicit operators

The condensed QP of equation (3.8) is the starting point for everything that follows. The next step is to factor its optimality conditions into a state-independent part, which can be computed once, and a state-dependent part, which has to be solved online.

3.3.1. Regularity and the inverse-action form

Assumption 3.1 (Regularity of the condensed QP)**.** For every state considered, (i) the input-cost weight satisfies $\boldsymbol{R} \succ 0$, so the condensed Hessian $\boldsymbol{H} \succ 0$ and the QP equation (3.8) is strictly convex with a unique primal minimiser $\boldsymbol{U}^\star(x_0)$; (ii) the state admits a feasible input sequence, so equation (3.8) is feasible; and (iii) at the optimal active set the active rows of $\boldsymbol{G}$ are linearly independent (the linear independence constraint qualification [34]), so the reduced block $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}$ of equation (3.15) is positive definite.

Strict convexity in Assumption 3.1(i) fixes a single primal optimiser per state. The multipliers and the active set do not need to be unique when constraints are degenerate, a distinction that the validation of section 5.4 relies on. At the solution of equation (3.8) the KKT stationarity condition reads

$$\boldsymbol{H}\boldsymbol{U}^\star + \boldsymbol{F}x_0 + \boldsymbol{G}^\top\boldsymbol{\lambda} = 0, \tag{3.9}$$

with multipliers $\boldsymbol{\lambda} \geq 0$ that are nonzero only on the active constraints [10], [35]. Because $\boldsymbol{H} \succ 0$ by Assumption 3.1(i), equation (3.9) can be solved for $\boldsymbol{U}^\star$ in terms of the state and the multipliers,

$$\boldsymbol{U}^\star = -\boldsymbol{H}^{-1}\boldsymbol{F}x_0 - \boldsymbol{H}^{-1}\boldsymbol{G}^\top\boldsymbol{\lambda} = -\boldsymbol{L}x_0 - \boldsymbol{M}\boldsymbol{\lambda}, \tag{3.10}$$

which separates the optimiser into a state-driven part and a multiplier-driven part. The two state-independent factors $\boldsymbol{L}$ and $\boldsymbol{M}$ are defined as the solutions of the linear systems

$$\boldsymbol{H}\,\boldsymbol{L} = \boldsymbol{F}, \qquad \boldsymbol{H}\,\boldsymbol{M} = \boldsymbol{G}^\top. \tag{3.11}$$

The notation $\boldsymbol{H}^{-1}$ in equation (3.10) is shorthand for the action of these solves and is never assembled explicitly. Both systems share the coefficient matrix $\boldsymbol{H}$, so a single Cholesky factorisation of $\boldsymbol{H}$ is computed once and reused, and each column of $\boldsymbol{L}$ and $\boldsymbol{M}$ follows by forward and backward substitution with that factor. This is cheaper and more stable than forming the inverse [35].

3.3.2. The state-independent operators

The matrix $\boldsymbol{L}$ maps the state to the unconstrained input sequence. When no constraint is active, $\boldsymbol{\lambda} = 0$ and equation (3.10) reduces to $\boldsymbol{U}^\star = -\boldsymbol{L}x_0$. The matrix $\boldsymbol{M}$ carries the correction needed when constraints are active. Substituting equation (3.10) into the active rows of $\boldsymbol{G}\boldsymbol{U} = \bar{\boldsymbol{w}} + \boldsymbol{E}x_0$ collapses the constraint into a system for $\boldsymbol{\lambda}$ that involves only two further state-independent quantities, the dual-coupling matrix and the constraint offset,

$$\boldsymbol{S} = \boldsymbol{G}\boldsymbol{M} = \boldsymbol{G}\boldsymbol{H}^{-1}\boldsymbol{G}^\top, \qquad \boldsymbol{T} = \boldsymbol{E} + \boldsymbol{G}\boldsymbol{L}. \tag{3.12}$$

The matrix $\boldsymbol{S}$ is symmetric positive semidefinite (and positive definite on the active rows under Assumption 3.1(iii)). It governs the reduced dual problem of section 3.4 [36]. The five operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}, \bar{\boldsymbol{w}}$ are state-independent. Each is one global matrix, computed once and reused at every state, and each is the object chapter 4 represents as a TT or TTM.

Table 3.1 collects these operators with their dimensions, the identities that construct them, and the roles they play in the online evaluator. The two state-mapping operators $\boldsymbol{L}$ and $\boldsymbol{T}$ have n columns, the two multiplier-coupled operators $\boldsymbol{M}$ and $\boldsymbol{S}$ have Np columns, and $\bar{\boldsymbol{w}}$ is a single column. The row count is Nm for the input-valued operators and Np for the constraint-valued ones. These shapes determine the storage of section 3.6 and the tensor factorisation of chapter 4.

Table 3.1: The semi-explicit operators: dimension, construction identity, and online role. All five are state-independent and computed once.

Operator	Size	Construction	Online role
$\boldsymbol{L}$	$Nm \times n$	solve $\boldsymbol{H}\boldsymbol{L} = \boldsymbol{F}$	unconstrained law $\boldsymbol{U}^{(0)} = -\boldsymbol{L}x_0$
$\boldsymbol{M}$	$Nm \times Np$	solve $\boldsymbol{H}\boldsymbol{M} = \boldsymbol{G}^\top$	primal correction $-\boldsymbol{M}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$
$\boldsymbol{S}$	$Np \times Np$	$\boldsymbol{S} = \boldsymbol{G}\boldsymbol{M}$	reduced dual solve $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{v}_{\mathcal{A}}$
$\boldsymbol{T}$	$Np \times n$	$\boldsymbol{T} = \boldsymbol{E} + \boldsymbol{G}\boldsymbol{L}$	residual base $\boldsymbol{v} = -(\bar{\boldsymbol{w}} + \boldsymbol{T}x_0)$
$\bar{\boldsymbol{w}}$	Np	$\bar{\boldsymbol{w}} = [\boldsymbol{w}^\top, \dots, \boldsymbol{w}^\top]^\top$	constraint offset in $\boldsymbol{v}$

3.3.3. The boundary of precomputation

The operators of equations (3.11) and (3.12) are the complete set of quantities the controller can precompute. They depend only on the system, cost, and constraint data, not on the state x_0 or on which constraints are active. The one quantity that cannot be precomputed is the multiplier vector $\boldsymbol{\lambda}$ in equation (3.10). By complementary slackness, $\boldsymbol{\lambda}$ has nonzero entries only on the active constraints, so determining it amounts to identifying which constraints are active. This active set $\mathcal{A}$ depends on x_0, and precomputing it for every x_0 would require enumerating a control law for every possible $\mathcal{A}$, the route taken by explicit MPC and the source of its combinatorial growth in storage [6], [7]. SE MPC stops at this boundary. What remains online is the small active-set problem of section 3.4, identifying the active constraints, extracting the corresponding operator blocks, and solving the reduced dual system.

3.3.4. Offline construction

The SE pipeline has two stages, an offline stage that builds the operators of table 3.1 and an online stage that uses them to solve the active-set problem at each state. The offline stage is given in Algorithm 2. The condensed Hessian $\boldsymbol{H}$ is factored once by Cholesky, since $\boldsymbol{H} \succ 0$, and the same factorisation is reused for the two solves in equation (3.11). The solve $\boldsymbol{H}\boldsymbol{M} = \boldsymbol{G}^\top$ carries one RHS per constraint and dominates the offline cost [10]. The matrices $\boldsymbol{S}$ and $\boldsymbol{T}$ are then formed by two matrix products.

Algorithm 2 Construction of the semi-explicit operators in matrix form [10].

Require: condensed operators $\boldsymbol{H}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{E}, \bar{\boldsymbol{w}}$
Ensure: $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}, \bar{\boldsymbol{w}}$
1: factor $\boldsymbol{H} = \boldsymbol{R}_\mathrm{c}^\top \boldsymbol{R}_\mathrm{c}$
2: solve $\boldsymbol{H}\,\boldsymbol{L} = \boldsymbol{F}$
3: solve $\boldsymbol{H}\,\boldsymbol{M} = \boldsymbol{G}^\top$
4: $\boldsymbol{S} \leftarrow \boldsymbol{G}\boldsymbol{M}, \quad \boldsymbol{T} \leftarrow \boldsymbol{E} + \boldsymbol{G}\boldsymbol{L}$

3.4. The active-set evaluator

The online stage is a dual active-set method [10], [35], [36], shown in Algorithm 3. The algorithm maintains a working set $\mathcal{A}$ of constraints held as equalities and updates it iteratively. At convergence, the working set contains exactly the constraints active at the optimum $\boldsymbol{U}^\star$, the optimal active set to which Assumption 3.1(iii) applies.

Define the constraint residual

$$\boldsymbol{g}(\boldsymbol{U}, x_0) = \boldsymbol{G}\boldsymbol{U} - \bar{\boldsymbol{w}} - \boldsymbol{E}x_0, \tag{3.13}$$

so that $\boldsymbol{U}$ is feasible exactly when $\boldsymbol{g}(\boldsymbol{U}, x_0) \leq 0$. The evaluator starts from the unconstrained optimum $\boldsymbol{U}^{(0)} = -\boldsymbol{L}x_0$, whose residual reduces, using $\boldsymbol{T} = \boldsymbol{E} + \boldsymbol{G}\boldsymbol{L}$, to

$$\boldsymbol{v} = -(\bar{\boldsymbol{w}} + \boldsymbol{T}x_0). \tag{3.14}$$

At each iteration, let $\boldsymbol{v}_\mathcal{A}$ denote the entries of $\boldsymbol{v}$ on the working set, and let $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}$ denote the block of $\boldsymbol{S}$ whose rows and columns are both indexed by $\mathcal{A}$. The working-set constraints are enforced as equalities by solving the reduced dual system

$$\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\,\boldsymbol{\lambda}_\mathcal{A} = \boldsymbol{v}_\mathcal{A}. \tag{3.15}$$

Under Assumption 3.1(iii), $\boldsymbol{S}_{\mathcal{A}\mathcal{A}} \succ 0$, so equation (3.15) is a small SPD solve whose size is the working-set size $|\mathcal{A}|$ rather than the full constraint count. The algorithm updates $\mathcal{A}$ by adding the most violated inactive constraint and dropping any constraint whose multiplier turns negative, and terminates when no inactive constraint is violated and every multiplier is nonnegative.

Algorithm 3 Semi-explicit active-set evaluation [10], [36].

Require: operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}, \bar{\boldsymbol{w}}$, state x_0
Ensure: optimal input sequence $\boldsymbol{U}^\star$
$\boldsymbol{U}^{(0)} \leftarrow -\boldsymbol{L}x_0, \quad \boldsymbol{v} \leftarrow -(\bar{\boldsymbol{w}} + \boldsymbol{T}x_0), \quad \mathcal{A} \leftarrow \varnothing$
if $\max_i v_i \le 0$ **then**
  return $\boldsymbol{U}^{(0)}$
end if
repeat
  solve $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{v}_{\mathcal{A}}$
  if $\min_i(\lambda_{\mathcal{A}})_i < 0$ **then**
    remove arg min index from $\mathcal{A}$; **continue**
  end if
  $\boldsymbol{r} \leftarrow \boldsymbol{v} - \boldsymbol{S}_{:,\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}}$
  $j \leftarrow \arg\max_{i \notin \mathcal{A}} r_i$
  if $r_j \le 0$ **then break**
  end if
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{j\}$
until converged
$\boldsymbol{U}^\star \leftarrow -\boldsymbol{L}x_0 - \boldsymbol{M}_{:,\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}}$

3.5. The global-operator variant

The only departure from the standard pipeline is in how the operators are stored. The per-active-set construction of Borrelli et al. [10] enumerates the reduced blocks $(\boldsymbol{S}_{\mathcal{A}\mathcal{A}}, \boldsymbol{T}_{\mathcal{A}})$ for each possible active set and stores them separately. This thesis takes the opposite choice. The five global operators of table 3.1 are kept in full, and the active block is extracted online by row selection,

$$\boldsymbol{S}_{\mathcal{A}\mathcal{A}} = \boldsymbol{\Sigma}_{\mathcal{A}}\,\boldsymbol{S}\,\boldsymbol{\Sigma}_{\mathcal{A}}^{\top}, \tag{3.16}$$

with $\boldsymbol{\Sigma}_{\mathcal{A}}$ the indicator of the active rows. Each precomputed object stays a single global matrix, which is what allows it to be represented as a TT in chapter 4.

The condensation, the KKT decomposition, and the active-set evaluator are standard [4], [6], [10]. They provide the reference pipeline that chapter 5 measures every compressed controller against. The adjustment made in this chapter is to store the operators as single matrices shared across all states, which is what chapter 4 compresses.

3.6. Storage and latency of the reference pipeline

The reference pipeline avoids both the per-step QP solve of online MPC and the enumerated partition of explicit MPC, at the price of the size of its global operators. Three costs of the matrix-form pipeline make that price precise and set the targets for chapter 4: the storage of the operator set, the offline cost of building it, and the online cost of evaluating it.

3.6.1. Storage

Summing the dimensions in table 3.1, the matrix operator set holds

$$B_{\text{mat}} = \underbrace{(Nm)n}_{\boldsymbol{L}} + \underbrace{(Nm)(Np)}_{\boldsymbol{M}} + \underbrace{(Np)^2}_{\boldsymbol{S}} + \underbrace{(Np)n}_{\boldsymbol{T}} + \underbrace{Np}_{\bar{\boldsymbol{w}}} \tag{3.17}$$

parameters. With the typical box-constraint count $p = 2(n+m)$, the principal dimension Np exceeds the input dimension Nm. The dual-coupling matrix $\boldsymbol{S}$ therefore dominates, quadratic in Np, with the inverse-action factor $\boldsymbol{M}$ one order lower. The remaining operators $\boldsymbol{L}, \boldsymbol{T}, \bar{\boldsymbol{w}}$ contribute little to the total. As the horizon or the constraint count grows, the memory ceiling of the matrix form is set by $\boldsymbol{S}$ [5], [20].

3.6.2. Offline construction

The cost of Algorithm 2 comes from a Cholesky factorisation of $\boldsymbol{H}$, two triangular solves (for $\boldsymbol{L}$ and $\boldsymbol{M}$), and two matrix products (for $\boldsymbol{S}$ and $\boldsymbol{T}$). Table 3.2 lists the cost of each step.

Table 3.2: Offline cost of Algorithm 2, in floating-point operations.

Step	Operation	Cost
Factorise $\boldsymbol{H}$	Cholesky	$\mathcal{O}\big((Nm)^3\big)$
Solve $\boldsymbol{HL} = \boldsymbol{F}$	n RHSs	$\mathcal{O}\big((Nm)^2 n\big)$
Solve $\boldsymbol{HM} = \boldsymbol{G}^\top$	Np RHSs	$\mathcal{O}\big((Nm)^2 Np\big)$
Form $\boldsymbol{S} = \boldsymbol{GM}$	dense product	$\mathcal{O}\big(Np^2\, Nm\big)$
Form $\boldsymbol{T} = \boldsymbol{E} + \boldsymbol{GL}$	dense product	$\mathcal{O}(Np\, Nm\, n)$

With $Np > Nm$, the multiple-RHS solve for $\boldsymbol{M}$ and the formation of $\boldsymbol{S}$ together set the offline budget. Both grow at least quadratically with Np [5], [20].

3.6.3. Online latency

Online, each iteration of Algorithm 3 adds or drops one constraint from $\mathcal{A}$ and the algorithm terminates when no inactive constraint is violated and every multiplier is nonnegative [4], [36]. Each iteration solves the reduced SPD system $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{v}_{\mathcal{A}}$ of size $|\mathcal{A}|$ and then updates the residual $\boldsymbol{r} = \boldsymbol{v} - \boldsymbol{S}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$ to pick the next candidate constraint. Table 3.3 lists the per-iteration cost.

Table 3.3: Per-iteration cost of Algorithm 3.

Step	Implementation	Cost
Reduced solve	Cholesky from scratch	$\mathcal{O}\big(\lvert\mathcal{A}\rvert^3\big)$
Reduced solve	incremental Cholesky update	$\mathcal{O}\big(\lvert\mathcal{A}\rvert^2\big)$
Residual update and candidate selection	touches full Np	$\mathcal{O}(Np\,\lvert\mathcal{A}\rvert)$

At small scale, $|\mathcal{A}|$ is small and the arithmetic is negligible. At scale, the limiting factor is not the floating-point work but the repeated memory access into the large operators $\boldsymbol{S}, \boldsymbol{M}, \boldsymbol{T}$, which are too large to fit in cache [11], [19], [20].

Chapter 4 addresses the three costs above by representing the operators of section 3.2.2 and section 3.3 in the TT format. Storage drops with the TT ranks of the tensorised operators. The offline cost is rebuilt on top of a tensor-native inverse-action solve. The online evaluator is recast as a sequence of TT contractions instead of matrix-form column fetches.

4

Tensorised Semi-Explicit MPC

Chapter 3 reduced the controller to five state-independent operators and identified three costs of storing and building them in matrix form. This chapter represents those operators as TTs and builds the construction and evaluation pipeline around them. The tensor-format tools used are standard: the TT-SVD and TT/TTM arithmetic [12] and the ALS [29]. What is new is how they are organised around the condensed SE operators: the index layout the operators share (section 4.2), the two construction routes, and the online evaluator. The inverse-action solve (section 4.7) is assembled from standard components, organised into the multiple-RHS solver that the construction of the SE operators requires.

4.1. Three tensorisation points

The reference pipeline of chapter 3 admits three points at which a TT representation can be introduced, each acting on a different object. The *late-stage* route compresses the final operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$ after they have been built by the reference pipeline (section 3.3). The *mid-stage* route tensorises the condensed operators $\boldsymbol{H}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{E}$ (section 3.2.2) and carries out the inverse-action construction in the TT format. The *early-stage* route tensorises the stacked matrices $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{Q}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{M}}_x, \bar{\boldsymbol{M}}_u$ (section 3.2.1) and performs the entire condensation in the TT format. The three routes share a common endpoint, the compressed operator set $\widetilde{\mathcal{L}}, \widetilde{\mathcal{M}}, \widetilde{\mathcal{S}}, \widetilde{\mathcal{T}}$ (with $\bar{\boldsymbol{w}}$ carried in vector form), and a common online evaluator (section 4.8), so they can be compared at the same interface. The vector $\bar{\boldsymbol{w}}$ contributes only Np entries to the storage count equation (3.17), negligible against the $(Np)^2$ of $\boldsymbol{S}$, so it is left uncompressed throughout.

4.2. Shape plans

To represent an operator as a TTM, its row and column indices must be split into modes by the tensorisation of Definition 2.5. The choice of split sets the core dimensions and TT ranks, and therefore the storage. The shape plan introduced here picks a different split for each operator, based on what that operator maps between.

Each operator's row and column dimensions admit a set of factorisations, and these factorisations determine which splits are possible. The dimensions themselves come from the stacking of chapter 3. The stacked input sequence $\boldsymbol{U} \in \mathbb{R}^{Nm}$ collects N input vectors of dimension m, the stacked constraints collect N blocks of p per-step constraints, and the current state is a single n-vector. A row or column index of a condensed operator therefore identifies one of the N time steps together with one component within that step, an input, a constraint, or a state entry. The Hessian $\boldsymbol{H} \in \mathbb{R}^{Nm \times Nm}$ maps one stacked input sequence to another, so each of its indices is a (time step, input) pair. The constraint matrix $\boldsymbol{G} \in \mathbb{R}^{Np \times Nm}$ pairs a (time step, constraint) row index with a (time step, input) column index.

Each index of an operator therefore has a temporal part, ranging over the horizon, and a physical part, ranging over one of three domains: inputs, constraints, or the current state. A *shape plan* assigns a temporal factor list and a physical factor list to the row index of an operator, and likewise to its column index. The temporal factor list $\tau = (N_1, \dots, N_{d_t})$ splits the horizon, $\prod_k N_k = N$. The physical factor lists μ, ν, ξ split

the input dimension m, the per-step constraint count p, and the state dimension n respectively. The current state has no temporal part, since it is a single vector rather than a per-step stack, so columns that map from it use the *identity temporal list* $\mathbb{1} = (1, \dots, 1)$ in place of τ. Concatenating the temporal and physical factors gives a TTM of order $d = d_t + d_x$, with d_t temporal cores and d_x physical cores.

Each row and column index splits into a temporal sub-index $(t_1, \dots, t_{d_t})$ and a physical sub-index $(\phi_1, \dots, \phi_{d_x})$ according to

$$t = \sum_{k=1}^{d_t} t_k \prod_{l<k} N_l, \qquad \phi = \sum_{k=1}^{d_x} \phi_k \prod_{l<k} a_l, \tag{4.1}$$

where $a = (a_1, \dots, a_{d_x})$ is the relevant physical factor list. The k-th core of the TTM carries the k-th temporal or physical index pair. The operator is then a chain of cores, temporal cores followed by physical cores, each with the row index on its upper leg and the column index on its lower leg (figure 4.1).

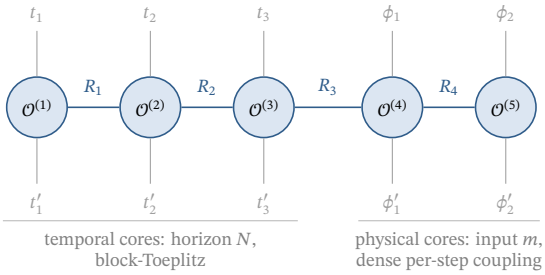


Figure 4.1: A TTM operator $\mathcal{O}$ under the plan $(\tau, \mu) \times (\tau, \mu)$, with $\tau = (N_1, N_2, N_3)$ and $\mu = (m_1, m_2)$. The $d = d_t + d_x$ cores $\mathcal{O}^{(k)}$ form a chain (equation (4.1)). Each upper leg is a row index (t_k or ϕ_k); each lower leg is the matching column index. Neighbouring cores are joined by TT ranks R_k.

The shape plan of each operator follows from its row and column domains (table 4.1). The Hessian $\boldsymbol{H}$ maps inputs to inputs, so both sides use (τ, μ); the dual-coupling matrix $\boldsymbol{S}$ maps constraints to constraints, so both sides use (τ, ν); the operators that act on the current state, $\boldsymbol{F}, \boldsymbol{E}, \boldsymbol{L}, \boldsymbol{T}$, use the identity temporal list on their state side.

Table 4.1: Shape plans of the condensed and semi-explicit operators. The temporal list τ splits the horizon; $\mathbb{1}$ is the identity temporal list for the current state; μ, ν, ξ split the input, constraint, and state dimensions.

Operator	Matrix-form size	Row plan	Column plan	Used in
$\boldsymbol{H}$	$Nm \times Nm$	(τ, μ)	(τ, μ)	inverse-action solve
$\boldsymbol{G}$	$Np \times Nm$	(τ, ν)	(τ, μ)	constraint map
$\boldsymbol{F}$	$Nm \times n$	(τ, μ)	$(\mathbb{1}, \xi)$	solve for $\boldsymbol{L}$
$\boldsymbol{E}$	$Np \times n$	(τ, ν)	$(\mathbb{1}, \xi)$	constraint shift
$\boldsymbol{L}$	$Nm \times n$	(τ, μ)	$(\mathbb{1}, \xi)$	unconstrained law
$\boldsymbol{M}$	$Nm \times Np$	(τ, μ)	(τ, ν)	multiplier correction
$\boldsymbol{S}$	$Np \times Np$	(τ, ν)	(τ, ν)	active-set dual system
$\boldsymbol{T}$	$Np \times n$	(τ, ν)	$(\mathbb{1}, \xi)$	residual evaluation

Each choice of factor lists imposes a cap on how large the TT ranks can grow. By equation (2.2), the TT rank at position k is bounded by whichever side of the cut is smaller, $\min\big(\prod_{l \le k} g_l, \prod_{l > k} g_l\big)$. The maximum rank along the cores is bounded by the worst case of this over all cuts. This worst case is the *unfolding exposure*

$$e(g) = \max_{1 \le k < d} \min\Big(\prod_{l \le k} g_l, \prod_{l > k} g_l\Big), \tag{4.2}$$

so that $R_{\max} \le e(g)$. The bound restates equation (2.2). Algorithm 4 uses $e(g)$ as the objective and minimises it jointly across operators that share a domain by searching the orderings of the prime factors of n, m, p. Each operator gets its own split rather than one shared across the set. Chapter 5 measures to what extent this choice matters for storage.

Algorithm 4 Operator-aware shape-plan construction.

Require: horizon N, dimensions n, m, p
Ensure: temporal list τ, physical lists μ, ν, ξ, operator plans
1: $\tau \leftarrow$ prime factorisation of N, with d_t the number of prime factors
2: enumerate the prime-factor groupings of n, m, p across all depths d_x
3: for each candidate (μ, ν, ξ), score the unfolding exposure of $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$
4: select the candidate of least worst-case exposure
5: assign each operator its row/column plan from table 4.1

4.3. Late-stage route

The late-stage route compresses the final operators directly. It builds $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$ by the reference pipeline (Algorithm 2) and applies the TT-SVD under the shape plans of table 4.1, carrying $\bar{\boldsymbol{w}}$ in vector form (Algorithm 5). This route compresses matrices that have already been built explicitly, isolating how compressible the final operators are independently of construction cost. The truncation accuracy is controlled by the bound of Algorithm 1 [12].

Algorithm 5 Late-stage tensorisation (adapted from the TT-SVD [12]).

Require: matrix-form $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$, shape plans, tolerance ε
Ensure: compressed $\widetilde{\mathcal{L}}, \widetilde{\mathcal{M}}, \widetilde{\mathcal{S}}, \widetilde{\mathcal{T}}$
1: **for** each operator $\boldsymbol{O} \in \{\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}\}$ **do**
2: reshape $\boldsymbol{O}$ to the modes of its shape plan
3: $\widetilde{\mathcal{O}} \leftarrow$ TT-SVD$(\boldsymbol{O}, \varepsilon)$
4: **end for**

4.4. Mid-stage route

The mid-stage route builds the compressed operators in TT form. It starts from the condensed operators $\boldsymbol{H}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{E}$, tensorises them through their Kronecker-sum structure, and passes the result to the inverse-action solve of section 4.6.

4.4.1. Target and structure

Each condensed operator is a sum of Kronecker products. These products separate time from state. Each term contains a temporal factor acting across the horizon and a physical factor acting within a single step. Tensorising the operator requires only tensorising these two small factors. This separation is the structure the TT format compresses.

A down-shift matrix explicitly defines this temporal separation. Let $\boldsymbol{J} \in \mathbb{R}^{N \times N}$ be the down-shift matrix, placing ones on the first subdiagonal:

$$\boldsymbol{J} = \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}, \qquad (\boldsymbol{J}^p)_{st} = \begin{cases} 1 & s = t + p, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

The matrix power $\boldsymbol{J}^p$ places ones on the p-th subdiagonal. When applied to a stacked horizon vector, $\boldsymbol{J}^p$ shifts the vector down by p steps. By definition, $\boldsymbol{J}^0 = \boldsymbol{I}_N$ and $\boldsymbol{J}^N = 0$. Using this notation, the stacked prediction matrix from section 3.2.1 becomes a temporal-shift sum:

$$\bar{\boldsymbol{B}} = \sum_{p=0}^{N-1} \boldsymbol{J}^p \otimes \boldsymbol{A}^p \boldsymbol{B}. \tag{4.4}$$

The stacked weight matrices apply the stage weight to every step. A terminal correction is then added to enforce the terminal weight on the final step:

$$\bar{\boldsymbol{Q}} = \boldsymbol{I}_N \otimes \boldsymbol{Q} + \left(\boldsymbol{e}_N \boldsymbol{e}_N^\top\right) \otimes (\boldsymbol{P} - \boldsymbol{Q}), \qquad \bar{\boldsymbol{R}} = \boldsymbol{I}_N \otimes \boldsymbol{R}. \tag{4.5}$$

Under the stage-weight convention of equation (4.5), the condensed Hessian of equation (3.6) takes the temporal-shift Kronecker-sum form:

$$\boldsymbol{H} = 2 \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\boldsymbol{J}^i)^\top \boldsymbol{J}^j \otimes \boldsymbol{B}^\top (\boldsymbol{A}^\top)^i \boldsymbol{Q}\, \boldsymbol{A}^j \boldsymbol{B} \;+\; 2\boldsymbol{I}_N \otimes \boldsymbol{R}. \tag{4.6}$$

This expression excludes the terminal correction. Applying the correction $\boldsymbol{P} - \boldsymbol{Q}$ adds one additional term of the same form for the final stage. The operators $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{E}$ also take this temporal-shift form.

To derive equation (4.6), substitute the stacked prediction from equation (4.4) into the Hessian definition $\boldsymbol{H} = 2(\bar{\boldsymbol{B}}^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{B}} + \bar{\boldsymbol{R}})$. Next, apply the mixed-product rule $(\boldsymbol{X} \otimes \boldsymbol{Y})(\boldsymbol{V} \otimes \boldsymbol{W}) = \boldsymbol{XV} \otimes \boldsymbol{YW}$ to each Kronecker factor. This substitution yields a double sum over the shift powers i and j. The double sum explicitly separates the temporal factor $(\boldsymbol{J}^i)^\top \boldsymbol{J}^j$ from the physical block $\boldsymbol{B}^\top (\boldsymbol{A}^\top)^i \boldsymbol{Q}\, \boldsymbol{A}^j \boldsymbol{B}$, resulting in equation (4.6). Finally, substituting equation (4.4) into the condensation identities from equations (3.6) and (3.7) expresses $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{E}$ in the same temporal-shift form.

The remaining three operators follow this same temporal-shift structure. Let $\boldsymbol{e}_t$ denote the t-th standard basis vector:

$$\boldsymbol{G} = \sum_{p=0}^{N-1} \boldsymbol{J}^p \otimes \boldsymbol{M}_x \boldsymbol{A}^p \boldsymbol{B} \;+\; \boldsymbol{I}_N \otimes \boldsymbol{M}_u, \qquad \boldsymbol{E} = -\sum_{t=1}^{N} \boldsymbol{e}_t \otimes \boldsymbol{M}_x \boldsymbol{A}^t, \qquad \boldsymbol{F} = 2 \sum_{i=0}^{N-1} \sum_{t=1}^{N} \left[(\boldsymbol{J}^i)^\top \boldsymbol{e}_t\right] \otimes \boldsymbol{B}^\top (\boldsymbol{A}^\top)^i \boldsymbol{Q}\, \boldsymbol{A}^t. \tag{4.7}$$

Like $\boldsymbol{H}$, the constraint map $\boldsymbol{G}$ carries a full horizon index $\boldsymbol{J}^p$. By contrast, the state-mapping operators $\boldsymbol{E}$ and $\boldsymbol{F}$ map directly from the single current state. Consequently, their column horizon collapses to a single index represented by the vector $\boldsymbol{e}_t$. This vector corresponds exactly to the identity temporal list defined in their shape plan (table 4.1). As before, each individual term explicitly separates into a temporal factor (containing $\boldsymbol{J}$ and $\boldsymbol{e}_t$) and a physical factor (containing $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{Q}$, $\boldsymbol{M}_x$, and $\boldsymbol{M}_u$).

The block-Toeplitz structure underlying equation (4.6) is well known. Prior work uses it for preconditioner design [37], and shift-matrix Kronecker sums are a standard way to write such structure [38]. The contribution here is the use of this form in the pipeline. Each term separates into a temporal and a physical factor, the shape plan maps the two onto cores, and the construction routes assemble the operators term by term in TT form.

Consider an individual term $(\boldsymbol{J}^i)^\top \boldsymbol{J}^j \otimes \boldsymbol{P}_{ij}$, where $\boldsymbol{P}_{ij} = \boldsymbol{B}^\top (\boldsymbol{A}^\top)^i \boldsymbol{Q}\, \boldsymbol{A}^j \boldsymbol{B}$. The shape plan maps the temporal factor to the temporal core and the physical block to the physical core. A TT rank of one joins these two cores. Before summation, each term forms a rank-one TTM. This represents the Kronecker product from figure 2.5, redrawn in figure 4.2.

The Hessian is the sum of these terms. The block-Toeplitz structure makes $\boldsymbol{P}_{ij}$ depend only on $i - j$, so equal-offset terms group together and the number K of distinct rank-one terms scales as $\mathcal{O}(N)$ rather than N^2. Summing K rank-one TTMs produces a new TTM with a maximum TT rank of K. This rank also cannot exceed the external dimensions of the connecting cores, N^2 for the vectorised temporal core (its $N \times N$ legs in figure 4.2) and m^2 for the physical core. Because $K = \mathcal{O}(N)$ is the smaller of the temporal quantities, the rank before any compression is bounded by:

$$r \;\leq\; \min\left(K,\; m^2\right). \tag{4.8}$$

This grouping by offset is the exploitable structure the route uncovers, the horizon coupling reduced to a small set of distinct temporal shifts. Rounding then reduces r further, achieving the much smaller rank permitted by the geometric decay detailed in section 4.10.

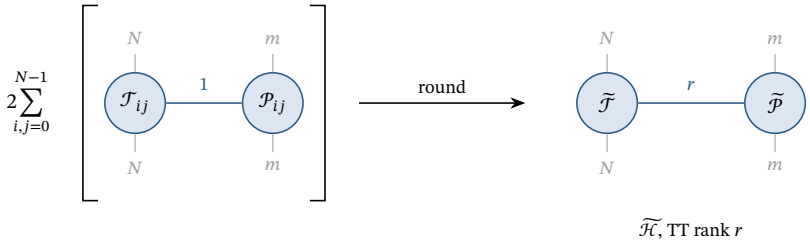


Figure 4.2: Mid-stage construction of the compressed Hessian $\widetilde{\mathcal{H}}$. Each summand in the Kronecker sum from equation (4.6) forms a rank-one TTM. This TTM consists of a temporal core $\mathcal{T}_{ij} = (\boldsymbol{J}^i)^\top \boldsymbol{J}^j$ and a physical core $\mathcal{P}_{ij} = \boldsymbol{B}^\top (\boldsymbol{A}^\top)^i \boldsymbol{Q}\, \boldsymbol{A}^j \boldsymbol{B}$, joined by a rank of one. In the diagram, the upper free legs carry the row indices, while the lower free legs carry the column indices. Each summand enters with TT rank one, summing over i and j raises the rank to at most K, the number of distinct terms, and rounding compresses the result into $\widetilde{\mathcal{H}}$, a final two-core TTM with TT rank r.

4.4.2. Assembly, cost, and an expected bottleneck

The assembly process consists of three steps. First, convert each Kronecker term to a TTM. Next, sum the terms together. Finally, round the result. These summation and rounding operations rely on the standard TTM arithmetic detailed in section 2.5 [12]. Once constructed, the compressed operators $\widetilde{\mathcal{H}}$, $\widetilde{\mathcal{F}}$, $\widetilde{\mathcal{G}}$, and $\widetilde{\mathcal{E}}$ proceed to the inverse-action solve in section 4.6.

The Hessian sum contains $K = \mathcal{O}(N)$ distinct Kronecker terms, and each term carries an $N \times N$ temporal factor of N^2 entries, so enumerating them costs $\mathcal{O}(N^3)$ work. This mid-stage enumeration becomes expensive at long horizons, creating an expected bottleneck in the pipeline and motivating the development of a second tensor-native constructor.

4.5. Early-stage route

The early-stage route moves the tensorisation earlier in the pipeline, attempting to reduce construction cost at long horizons. It first tensorises the stacked matrices and then performs the entire condensation directly in the TT format.

4.5.1. Target and structure

The route targets the stacked matrices $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{Q}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{M}}_x, \bar{\boldsymbol{M}}_u$ of section 3.2.1. These have compact Kronecker forms with the same temporal-shift structure the mid-stage route uncovered (section 4.4.1). $\bar{\boldsymbol{B}}$ is the N-term temporal-shift sum equation (4.4), and the weights and constraint maps are single Kronecker products, $\bar{\boldsymbol{Q}} = \boldsymbol{I}_N \otimes \boldsymbol{Q}$, $\bar{\boldsymbol{R}} = \boldsymbol{I}_N \otimes \boldsymbol{R}$, $\bar{\boldsymbol{M}}_x = \boldsymbol{I}_N \otimes \boldsymbol{M}_x$, and $\bar{\boldsymbol{M}}_u = \boldsymbol{I}_N \otimes \boldsymbol{M}_u$. Each tensorises directly under its shape plan. Working from the primitives avoids the $\mathcal{O}(N^3)$ enumeration cost of the mid-stage Hessian equation (4.6), though the construction cost moves into the TTM products and rounding of the condensation itself, quantified below.

4.5.2. TT-native condensation

The condensed operators are assembled directly from the compressed primitives by the condensation identities of section 3.2.2, evaluated as TTM products and sums with a rounding after every operation:

$$\widetilde{\mathcal{H}} = \mathrm{round}\big(2(\bar{\mathcal{B}}^\top \bar{\mathcal{Q}}\, \bar{\mathcal{B}} + \bar{\mathcal{R}})\big), \tag{4.9}$$

$$\widetilde{\mathcal{F}} = \mathrm{round}\big(2\, \bar{\mathcal{B}}^\top \bar{\mathcal{Q}}\, \bar{\mathcal{A}}\big), \tag{4.10}$$

$$\widetilde{\mathcal{G}} = \mathrm{round}\big(\bar{\mathcal{M}}_x\, \bar{\mathcal{B}} + \bar{\mathcal{M}}_u\big), \tag{4.11}$$

$$\widetilde{\mathcal{E}} = \mathrm{round}\big(-\bar{\mathcal{M}}_x\, \bar{\mathcal{A}}\big). \tag{4.12}$$

Each operator therefore costs at most three tensor-native operations: $\widetilde{\mathcal{H}}$ needs two products (first $\bar{\boldsymbol{B}}^\top \bar{\boldsymbol{Q}}$, then that times $\bar{\boldsymbol{B}}$) and one addition; $\widetilde{\mathcal{F}}$ needs two products; $\widetilde{\mathcal{G}}$ needs one product and one addition; and $\widetilde{\mathcal{E}}$ is a single product. Every product must respect mode compatibility, the column factors of the left operand matching the row factors of the right, which the shape plans of table 4.1 guarantee. Without rounding

after each operation the ranks would compound through the cores (quantified below). Figure 4.3 draws the condensation identities as tensor networks.

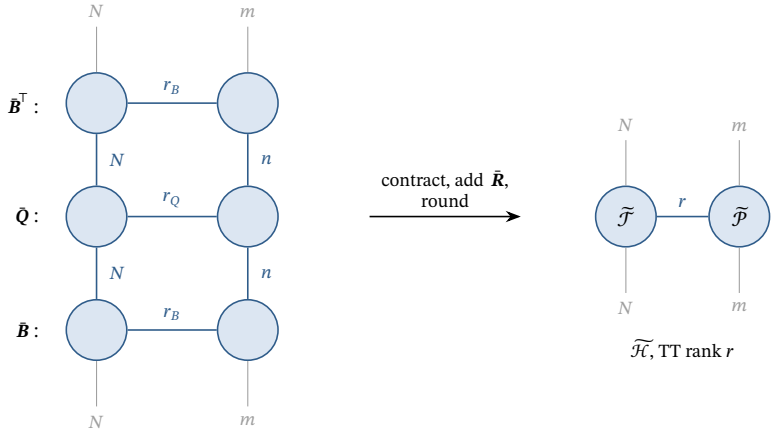


Figure 4.3: Early-stage construction of $\widetilde{\mathcal{H}}$ by native TTM products of the compressed primitives. Each primitive is a two-core TTM: a temporal core and a physical core joined by a horizontal edge of rank r_B or r_Q. The product $\bar{\mathcal{B}}^\top \bar{\mathcal{Q}}\, \bar{\mathcal{B}}$ is contracted mode by mode. The vertical edges carry the shared horizon dimension N in the temporal column and the state dimension n in the physical column; they connect the column legs of each TTM to the row legs of the next. The surviving outer legs are the row and column indices of the product. Adding $\bar{\mathcal{R}}$, rounding, and collapsing the cores into $\widetilde{\mathcal{H}}$ gives the same two-core TTM produced by the mid-stage construction (figure 4.2).

4.5.3. Cost and expected bottleneck

The early-stage route replaces the cubic temporal enumeration of the mid-stage with a fixed sequence of TTM products, so its pre-solve cost grows more slowly in the horizon.

Rounding follows every operation, but this does not prevent intermediate rank growth. Contracting $\bar{\mathcal{B}}^\top$ with $\bar{\mathcal{Q}}$ and then with $\bar{\mathcal{B}}$ reaches a rank of order $r_B^2\, r_Q$ before the next round truncates it. The weights $\bar{\mathcal{Q}}$ are a sum of two Kronecker products equation (4.5), so $r_Q \leq 2$. The lifted dynamics rank r_B grows with the horizon, so the peak rank is roughly the square of the final rank r_H. Rounding cost scales cubically in the TT rank, so this peak dominates construction cost. The inverse-action solve of section 4.6 incurs similar growth. Chapter 5 measures which is larger.

Whichever route is taken, the output is the same four condensed operators in TT form. Each follows the anatomy of figure 4.1, temporal cores followed by physical cores, with the row and column indices listed in table 4.1.

4.6. The inverse-action problem in TT format

Both construction routes reach the same two systems, now in the TT format:

$$\widetilde{\mathcal{H}}\,\widetilde{\mathcal{L}} = \widetilde{\mathcal{F}}, \qquad \widetilde{\mathcal{H}}\,\widetilde{\mathcal{M}} = \widetilde{\mathcal{G}}^\top, \tag{4.13}$$

with the compressed Hessian, right-hand sides, and unknowns all TT objects. The first system carries the n columns of the state-to-input gain, held by the state factor list ξ. The second carries Np columns, one per constraint, which makes $\widetilde{\mathcal{M}}$ the multiple-RHS inverse action of section 3.6.

Figure 4.4 draws the two systems as tensor networks. In each system, the operator and the unknown are contracted along their shared (τ, μ) index, the vertical edges between the two chains. The legs that remain open are the external indices of the result, and they match the RHS.

Neither system is solved by forming $\widetilde{\mathcal{H}}^{-1}$. The inverse exists since $\widetilde{\mathcal{H}} \succ 0$, but its TT representation generally has much larger ranks than the operator itself [24]. Storing it would undo the compression the rest of the pipeline depends on. The systems are solved iteratively, one core at a time. At each step the solver fixes every core of the unknown ($\widetilde{\mathcal{L}}^{(1)}, \ldots, \widetilde{\mathcal{L}}^{(d)}$ or $\widetilde{\mathcal{M}}^{(1)}, \ldots, \widetilde{\mathcal{M}}^{(d)}$) except one and minimises the energy over that single core. This reduces to a small linear system in the active core. A sweep moves through the cores, updating

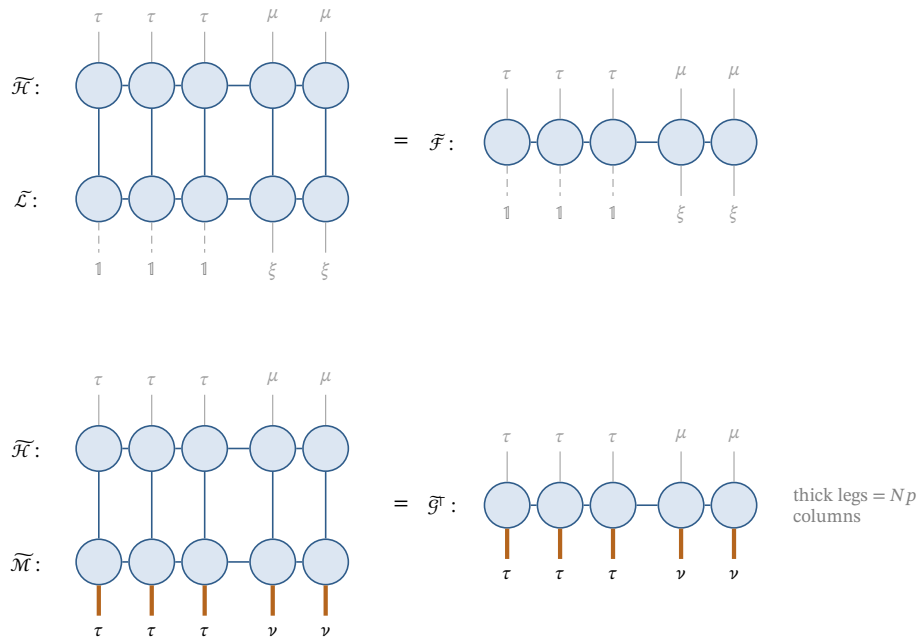


Figure 4.4: The two inverse-action systems equation (4.13) as tensor networks. Vertical edges mark the contracted modes. The open outer legs are the external indices of the result. Top: the system for $\widetilde{\mathcal{L}}$, whose column legs are the identity temporal list $\mathbb{1}$ and the state factors ξ. Bottom: the multiple columns of $\widetilde{\mathcal{M}}$, drawn thick along the constraint index ν; the solver of section 4.7 handles them in chunks.

each core in turn until the residual converges. This is the AMEn method of Dolgov and Savostyanov [30], which builds on ALS [29].

The first system $\widetilde{\mathcal{H}}\widetilde{\mathcal{L}} = \widetilde{\mathcal{F}}$ fits this setting directly. The second does not. $\widetilde{\mathcal{M}}$ is a TTM with Np columns, and solving them one at a time would discard the structure they share and repeat the sweep Np times. The solver of section 4.7 adapts the method to this multiple-RHS system.

4.7. A multiple-RHS SPD TT solver

The solver minimises the energy $\frac{1}{2}\langle\widetilde{\mathcal{X}}, \widetilde{\mathcal{H}}\widetilde{\mathcal{X}}\rangle - \langle\widetilde{\mathcal{X}}, \widetilde{\mathcal{B}}\rangle$, with $\widetilde{\mathcal{B}}$ equal to $\widetilde{\mathcal{F}}$ or $\widetilde{\mathcal{G}}^{\mathsf{T}}$. It combines four standard ingredients, listed in Algorithm 6, with one addition for the multiple RHSs. Simultaneous TT solution of multiple RHSs has been addressed before, by carrying the RHS index as an extra tensor mode and solving the enlarged system with a TT variant of GMRES [25]. The solver here stays instead in the energy-minimisation family, which exploits the positive-definiteness of $\widetilde{\mathcal{H}}$.

Fixing every core of $\widetilde{\mathcal{X}}$ except the k-th reduces the global system to a small local one,

$$\boldsymbol{H}_k \, \boldsymbol{x}_k = \boldsymbol{b}_k,$$

where $\boldsymbol{x}_k$ is the active core $\widetilde{\mathcal{X}}^{(k)}$ flattened to a vector. The *frame* $\boldsymbol{P}_k$ is the linear map that rebuilds the full tensor from this single core when the others are held fixed, $\widetilde{\mathcal{X}} = \boldsymbol{P}_k \, \boldsymbol{x}_k$. Its columns are the contractions of the surrounding cores, which is why it is called a frame. The local operator $\boldsymbol{H}_k = \boldsymbol{P}_k^{\mathsf{T}} \widetilde{\mathcal{H}} \boldsymbol{P}_k$ is the projection of $\widetilde{\mathcal{H}}$ onto this active core and inherits the positive-definiteness of $\widetilde{\mathcal{H}}$.

The simplest sweep is *one-site ALS*: solve the local system at each k, move to the next core, repeat [29]. It is cheap but cannot grow the ranks, so it fails when the solution rank is larger than the starting rank. The two-site variant MALS fixes this. Two neighbouring cores are merged into a single block, the merged local system is solved, and a truncated SVD splits the result back into two cores [29]. The truncation tolerance sets the new rank, so ranks adapt as the sweep proceeds.

Sweeps can still stagnate at a rank too small for the solution. Between sweeps the solver therefore takes one step of steepest descent on the residual,

$$\tfrac{1}{2}\|\widetilde{\mathcal{H}}\widetilde{\mathcal{X}} - \widetilde{\mathcal{B}}\|_F^2,$$

along the negative gradient $\widetilde{\mathcal{Z}} = \widetilde{\mathcal{H}}^\top(\widetilde{\mathcal{B}} - \widetilde{\mathcal{H}}\widetilde{\mathcal{X}})$. The step length is chosen to minimise the residual along $\widetilde{\mathcal{Z}}$. Along the line $\widetilde{\mathcal{X}} + \alpha\,\widetilde{\mathcal{Z}}$ the residual is quadratic in α, so the optimum is given by one division:

$$\alpha^\star = \frac{\|\widetilde{\mathcal{Z}}\|_F^2}{\|\widetilde{\mathcal{H}}\widetilde{\mathcal{Z}}\|_F^2}.$$

This is the *Cauchy step*, the exact line-search step for steepest descent on a quadratic [34]. After the update, the tensor is rounded back to manageable rank, which can erase the gain from the Cauchy step. A guard halves α and retries if the residual is larger after rounding than before. This enrichment plays the same role as the one in AMEn [30]. It injects residual information into the iterate, which lets the rank grow when sweeps stagnate. AMEn enriches inside the sweep. After each local solve, it appends a few residual-derived columns to the next core's basis. The following two-site SVD folds them into the iterate. This mechanism was designed for a single TT vector. Applying it to a TTM with Np columns either inflates every local block by a factor of Np, or enriches each column independently and abandons the joint TT representation of $\widetilde{\mathcal{M}}$. The solver here therefore enriches between sweeps, adding one scaled copy of the residual gradient $\widetilde{\mathcal{Z}}$ to the whole tensor. The same descent direction serves all Np columns at once, and the per-core work in the sweep stays untouched.

The local operator $\boldsymbol{H}_k$ is never stored as a matrix. Building it would mean contracting $\widetilde{\mathcal{H}}$ with the frame $\boldsymbol{P}_k$ on both sides, then writing the dense $n_k \times n_k$ result ($n_k = R_{k-1}\,I_k\,R_k$) to memory, resulting in $\mathcal{O}\big(R^4\,I^2\big)$ per core. The local system is therefore solved by *matrix-free preconditioned conjugate gradient (PCG)* [39], the standard iterative method for SPD systems, which needs only the product of $\boldsymbol{H}_k$ with a vector of the same size as $\boldsymbol{x}_k$ at each iteration. Each such product is a short tensor-network contraction: reshape the vector into a core with the shape of $\widetilde{\mathcal{X}}^{(k)}$, contract $\widetilde{\mathcal{H}}$ with the surrounding frame $\boldsymbol{P}_k$, then flatten the result back to a vector.

The matrix-free method removes the $\mathcal{O}\big(R^4\,I^2\big)$ matrix, but the iterate itself still grows with the right-hand-side count. Carrying all Np columns of $\widetilde{\mathcal{M}}$ through one sweep stores a block iterate of size $\mathcal{O}\big(Np\,R^2\,I\big)$ per core. Two-site merges push R up to the intermediate peak discussed in section 4.5.3 before truncation, so the worst memory is reached mid-sweep on the full column count.

The solver processes the columns in chunks of $c \ll Np$. Each chunk runs the matrix-free PCG solve on c columns [40]. Peak memory drops to $\mathcal{O}\big(c\,R^2\,I\big)$ per core. The operator is traversed Np/c times in return, increasing run time.

Chunking lowers peak memory without altering the converged $\widetilde{\mathcal{M}}$. Each column is its own SPD system with the same local operator $\boldsymbol{H}_k$, so its PCG convergence and final rank do not depend on the chunk it sits in. Columns inside one chunk still share the operator and frame contractions, so the joint structure is preserved within a chunk.

Three safeguards keep the sweep robust. The best iterate seen so far is kept across sweeps. A two-site update that worsens the residual is rejected. A final sweep at a tighter tolerance polishes the result.

4.8. The tensor-compatible online evaluator

The online controller runs the same active-set method as the matrix-form evaluator of chapter 3 (Algorithm 3). Only the operator actions change. Matrix-vector products are carried out as contractions directly on the cores, and the block and column extractions that the matrix form reads from stored arrays become index-selected contractions. The operators are never decompressed. These operation-level changes are listed in Algorithm 7.

The unconstrained candidate $-\widetilde{\mathcal{L}}\,x_0$ and the residual base $-\bar{\boldsymbol{w}} - \widetilde{\mathcal{T}}\,x_0$ are TTM–vector products. The state x_0 is reshaped to match the column legs of $\widetilde{\mathcal{L}}$ and $\widetilde{\mathcal{T}}$. The temporal column collapses to a single index $\mathbb{1}$, and the state dimension splits across the physical-core column legs (figure 4.1). The contraction then proceeds

Algorithm 6 Multiple-RHS SPD TT solve for $\widetilde{\mathcal{H}}\widetilde{\mathcal{X}} = \widetilde{\mathcal{B}}$ [29], [39], [40].

Require: SPD operator $\widetilde{\mathcal{H}}$, RHS $\widetilde{\mathcal{B}}$ (Np columns for $\widetilde{\mathcal{M}}$), tolerance ε
Ensure: $\widetilde{\mathcal{X}}$ with relative residual $\leq \varepsilon$
1: initialise $\widetilde{\mathcal{X}}$; record it as the best iterate
2: **repeat**
3: restore the best iterate if the previous sweep increased the residual
4: **enrichment:** $\widetilde{\mathcal{Z}} \leftarrow \widetilde{\mathcal{H}}^{\top}(\widetilde{\mathcal{B}} - \widetilde{\mathcal{H}}\widetilde{\mathcal{X}})$; $\widetilde{\mathcal{X}} \leftarrow \widetilde{\mathcal{X}} + \alpha\widetilde{\mathcal{Z}}$ with optimal step α; round
5: **one-site sweep:** for each core, solve the local SPD system by matrix-free PCG; round
6: **two-site sweep:** merge, solve, split by truncated SVD; reject if the residual grows; round
7: **until** relative residual $\leq \varepsilon$ or no progress
8: final polishing pass at a tighter local tolerance
Multiple-RHS: local PCG solves the Np columns in chunks to bound peak memory.

mode by mode, each operator core merging with the matching slice of x_0 and yielding a TT vector for the candidate sequence.

The reduced dual system $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{v}_{\mathcal{A}}$ of chapter 3 is kept in matrix form at the active-set size q. A $q \times q$ system is far smaller than the operators, and a tensor representation of it would cost more than it saves. The active block $\widetilde{\mathcal{S}}_{\mathcal{A}\mathcal{A}}$ is extracted from the TTM $\widetilde{\mathcal{S}}$ by index selection. The residual and primal corrections need products of the operators with a vector supported on the active set only, $\widetilde{\mathcal{S}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$ and $\widetilde{\mathcal{M}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$. These are computed by a selected-column contraction that visits only the active columns, grouping them by shared mode-index prefix so that the partial contractions common to several columns are computed once. A further optimisation applies because the receding-horizon law uses only the first input, meaning that only the first m rows of $\widetilde{\mathcal{L}}$ and $\widetilde{\mathcal{M}}$ are contracted, not the whole horizon.

Algorithm 7 Tensor-compatible semi-explicit active-set evaluation.

Require: TT operators $\widetilde{\mathcal{L}}, \widetilde{\mathcal{M}}, \widetilde{\mathcal{S}}, \widetilde{\mathcal{T}}$, vectors $\bar{\boldsymbol{w}}$ and x_0
Ensure: applied input $u_0^{\star}$
1: $\boldsymbol{U}^{(0)} \leftarrow -\widetilde{\mathcal{L}}x_0, \quad \boldsymbol{v} \leftarrow -(\bar{\boldsymbol{w}} + \widetilde{\mathcal{T}}x_0)$
2: run the active-set loop of Algorithm 3, with:
3: active block $\widetilde{\mathcal{S}}_{\mathcal{A}\mathcal{A}}$ by TTM index extraction; reduced solve kept in matrix form at size q
4: residual update $\boldsymbol{v} - \widetilde{\mathcal{S}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$ by selected-column contraction
5: recover $u_0^{\star}$ from the first m rows of $-\widetilde{\mathcal{L}}x_0 - \widetilde{\mathcal{M}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$

Access cost

Applying a TTM to a TT vector is a core-by-core contraction (section 2.5.3). The controller pairs the first core of the operator with the first core of the vector, then the next pair, and so on along the cores. The matrix operator is never formed. The arithmetic at each core is proportional to the size of that core, which equals the number of entries it stores. Storage and per-access cost therefore scale with the same per-core sum $\sum_k R_{k-1} I_k J_k R_k$. Reducing storage by any factor reduces the online work by the same factor, so a single TT-format choice controls both (equation (4.14)).

Let $\boldsymbol{O}$ be a TTM with cores of size $R_{k-1} \times I_k \times J_k \times R_k$, acting on a TT vector with TT ranks $(s_0, \dots, s_d)$. Its storage and the cost of one matrix–vector contraction are

$$B(\boldsymbol{O}) = \sum_k R_{k-1} I_k J_k R_k, \qquad C(\boldsymbol{O}) = \sum_k s_{k-1} s_k R_{k-1} I_k J_k R_k. \tag{4.14}$$

The access cost is the storage sum with each term reweighted by the vector ranks $s_{k-1} s_k$. For a fixed vector profile,

$$\min_k (s_{k-1}s_k)\, B(\boldsymbol{O}) \;\leq\; C(\boldsymbol{O}) \;\leq\; \max_k (s_{k-1}s_k)\, B(\boldsymbol{O}),$$

so C scales linearly with B. The contraction proceeds core by core, forming at core k a product core of size $(R_{k-1}s_{k-1}) \times I_k \times (R_k s_k)$ at cost $\mathcal{O}(s_{k-1}s_k R_{k-1} I_k J_k R_k)$ [12], and summing these per-core costs gives C.

This proportionality has a direct consequence for the online controller. Latency tracks the total parameter count $\sum_k R_{k-1} I_k J_k R_k$, not the maximum rank. Bounding every term by the maximum rank gives the loose estimate $\mathcal{O}\big(dR^2IJ\big)$. Equation (4.14) is the sharper statement, and section 5.9 confirms it empirically.

Equation (4.14) bounds the cost of contracting a full TTM with a TT vector, but the online loop rarely performs a full contraction. The selected-column matvecs $\widetilde{\mathcal{S}}_{:,\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}}$ and $\widetilde{\mathcal{M}}_{:,\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}}$ visit only the active columns, so their cost scales with the active-set size $|\mathcal{A}|$ rather than with the full column count. The reduced dual system $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\,\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{\upsilon}_{\mathcal{A}}$ is a small $q \times q$ matrix solve with $q = |\mathcal{A}|$, done outside the TT format. The online latency reported in chapter 5 therefore depends on the access cost and on $|\mathcal{A}|$, not on the maximum rank alone.

4.9. Approximation sources

The TT pipeline trades accuracy for compression.

Each route's accuracy is summarised by the relative error of the final operators. For route r and operator $\boldsymbol{O}$ the relative error is

$$e_{r,\boldsymbol{O}} = \frac{\|\widetilde{\mathcal{O}} - \boldsymbol{O}\|_F}{\|\boldsymbol{O}\|_F},$$

and $e_{r,\infty} = \max_{\boldsymbol{O}} e_{r,\boldsymbol{O}}$ is the worst case across operators.

Every intermediate tolerance is derived from a single target δ on the final-operator error. Let $\varepsilon_p, \varepsilon_c, \varepsilon_s, \varepsilon_a, \varepsilon_f$ be the tolerances for the primitive rounding, the initial compression, the inverse-action solve, the assembly, and the final rounding. The schedule sets

$$\varepsilon_f = \delta, \qquad \varepsilon_p \le \varepsilon_c \le \varepsilon_s \le \varepsilon_a \le \varepsilon_f, \qquad \varepsilon_\bullet \ge \varepsilon_{\text{mach}}, \tag{4.15}$$

with each stage at least as tight as the ones downstream, and each floored at the machine-precision level $\varepsilon_{\text{mach}}$, below which rounding cannot improve the result. The ordering keeps any upstream stage from contributing more error than δ allows, so the achieved error tracks δ.

The ratios between stages are tuned constants rather than derived quantities. Chapter 5 measures whether the achieved error actually tracks δ.

4.10. Predicted outcomes

This section predicts how the tensor controller will behave, ahead of the measurements in chapter 5. Each subsection states one hypothesis together with the structural reason behind it, and chapter 5 then tests it. All five predictions assume a stable plant, meaning every eigenvalue of the system matrix $\boldsymbol{A}$ is smaller than one in magnitude, so $|\lambda_E| < 1$ for the largest-magnitude eigenvalue λ_E. The undriven state then decays to zero over time. This is the case for the mass–spring–damper benchmark of chapter 5 because it is damped. For an unstable plant these predictions would not apply.

4.10.1. Rank scaling

The condensed Hessian $\boldsymbol{H}$ couples each pair of input steps in the prediction horizon through a power of $\boldsymbol{A}$ that grows with how far apart the two steps are. Because the plant is stable, those powers shrink geometrically, so blocks of $\boldsymbol{H}$ far from the diagonal are much smaller than blocks near it. The same decay is passed to $\boldsymbol{M}$ and $\boldsymbol{S}$, which are built from $\boldsymbol{H}$ through its inverse. Inverses inherit this decay, a classical result for banded matrices that extends to the Kronecker structure of $\boldsymbol{H}$ [41], [42].

This leads to a prediction for how rank trades off against accuracy. Let δ be the relative accuracy asked of the stored operator. The blocks shrink geometrically at each step away from the diagonal. To reach accuracy δ only the blocks larger than δ need to be kept. The rest can be discarded. Because the shrinkage is by a constant factor, the number of blocks worth keeping grows with the order of magnitude of $1/\delta$, which is equal to $\log(1/\delta)$. Each block kept adds a fixed amount of rank, so the maximum TT rank is expected to grow like $\log(1/\delta)$.

4.10.2. Storage

A TTM stores $\sum_k R_{k-1} I_k J_k R_k$ parameters (equation (2.4)). The shape plan (section 4.2) sets the mode sizes, so for a fixed shape plan only the ranks change with the target accuracy. The rank rule of the previous subsection then carries over to storage.

Bounding every rank by the largest rank R, each core costs at most R^2 times its mode size, so storage grows with the square of R. Since that rank is expected to grow only as $\log(1/\delta)$, while the matrix form grows with Np^2, the stored size is predicted to stay well below the matrix form, with the gap widening as the horizon grows. No such prediction holds along the system-size axis, because the physical dimensions n, m, p are fixed by the system rather than chosen, and the splits a shape plan can reach depend on how those sizes happen to factor.

4.10.3. Accuracy required for constraint satisfaction

Compression replaces each operator by an approximation with relative error δ, set by the requested accuracy. The cost and the constraints respond to this error differently. The solution sits at a minimum of the quadratic cost, where the cost is stationary, so an error in the cost operator moves the solution only slightly and raises the achieved cost by an amount of order δ^2. Near a constraint boundary, even a small error can flip which constraints the controller marks as active; the chosen input can then move just past the boundary and violate it. Constraint satisfaction is therefore lost before optimality, because the constraint error acts at first order while the cost error acts only at second order. This makes constraint satisfaction the tighter limit on δ, and aggressive compression can break it well before the operator error itself looks large.

The size of the violation depends on the conditioning of the active part of $\boldsymbol{S}$, the matrix the online solver inverts. The worse the conditioning, the more a small operator error is amplified [43]. Condensing builds this matrix from powers of $\boldsymbol{A}$ across the horizon. These powers span a wide range of magnitudes, from order one down to order $|\lambda_E|^N$, and the range widens as the horizon grows. The eigenvalues of the matrix spread with it, so its conditioning worsens with the horizon [37]. The accuracy needed to keep the constraints satisfied is therefore expected to tighten with the horizon.

4.10.4. Construction cost

Both routes build operators by TT sums and products. Each such operation makes the TT rank grow, and the rounding step that follows brings it back down. Rounding dominates the cost of these operations. It costs $\mathcal{O}\left(d\, I\, R^3\right)$, cubic in the TT rank R (section 2.5.5). The rank peaks just before each rounding step, above the rank of the finished operator, so it is this peak intermediate rank, not the final rank, that sets the cost.

The inverse-action solves dominate construction, and the multiple-RHS $\widetilde{\mathcal{M}}$-solve is the hardest because it carries all Np columns of $\widetilde{\mathcal{M}}$ at once. The solver sweeps along the cores and solves a small system at each core by matrix-free PCG (section 4.7), avoiding the dense $\mathcal{O}\left(R^4 I^2\right)$ local operator. Its total work is the cost of one local solve, times the number of cores, times the number of columns. The number of cores grows only logarithmically with the horizon ($\mathcal{O}(\log N)$ temporal cores plus a fixed number of physical cores), and the column count Np grows linearly. The rank, by contrast, enters at a much higher power. The local solves are polynomial in R, and the rounding step is cubic in it (section 2.5.5).

The rank is therefore what construction cost is most sensitive to. It is set by the target accuracy rather than chosen directly, while the number of cores and the column count are fixed overhead. For this reason, the construction time is expected to scale as a low power of the rank.

4.10.5. Online cost

Each control action contracts the stored operators against the current state. One contraction costs the per-core access sum of equation (4.14), $\sum_k R_{k-1} I_k J_k R_k$, the same sum that gives the storage. This cost grows with the TT ranks, so high ranks make each access expensive.

The tensor and matrix-form controllers run the same active-set loop and solve the same dual system at each step. They differ only in how they apply the operators. The matrix-form evaluator holds the full operators in memory, so applying them and extracting the active blocks as constraints enter and leave is just reading and multiplying stored entries, which it caches and reuses cheaply. The tensor evaluator holds only the cores, so it rebuilds each operator action by contracting them, paying the access cost at every step. That contraction visits one term per active column, so its cost grows with the active-set size. The tensor evaluator is therefore predicted to slow down relative to the matrix form as the active set grows.

5

Experimental Validation and Analysis

Chapter 4 developed the tensorised SE controllers, the shape plans, the construction routes, and the tensor-compatible evaluator. It closed by predicting where the TT representation should save memory and where its construction and online costs should dominate. This chapter answers the four supporting questions by testing those predictions against measurement. The operators are built and run on a mass–spring–damper benchmark, a classic linear control problem whose two size axes scale cleanly, so that each cost can be tracked as the problem grows. The experiments proceed in sequence, from a reference check of the matrix-form controller (section 5.4) through accuracy, storage, construction, and online cost (section 5.5–section 5.9).

The outcome is mixed. The compression measured in section 5.7 is large at admissible accuracy. The construction cost of section 5.8 and the online latency of section 5.9 do not improve on the matrix-form controller. Each shortfall is traced to a specific mechanism from chapter 4.

5.1. Experimental design

The six experiments form a dependency chain (figure 5.1). E0 checks that the matrix-form SE controller reproduces a conventional solver, so it can serve as the reference. E1 checks that the requested accuracy δ behaves as a usable parameter. E2 finds the loosest accuracy at which the compressed controller remains admissible in closed loop. E3, E4, and E5 then measure the three costs: storage, construction, and online latency. Using the same reference and admissible accuracy throughout makes the three cost claims comparable.

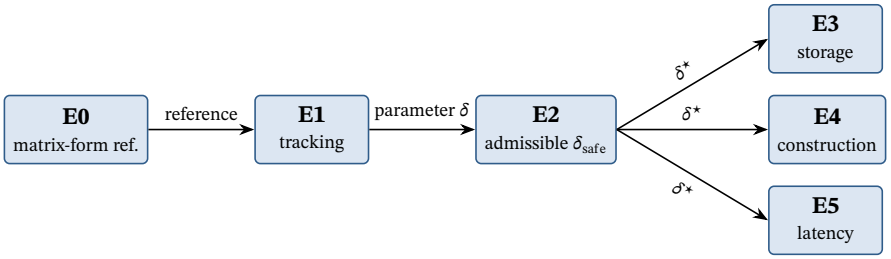


Figure 5.1: The experiment dependency chain. Each experiment supplies one piece of evidence and hands its conclusion to the next.

Two decisions cut redundant runs and keep the cost numbers comparable. First, E1 shows that the late, mid, and early routes all reach the same final-operator error at a given δ. The late route can therefore stand in for the other two in E2 and E3, and the three are compared directly only in E4, where their construction cost differs. Second, the accuracy is fixed once rather than tuned per case. E2 returns an admissible accuracy δ_{safe} that depends on the configuration and the workload, from about 10^{-4} in the mildest cases to about 10^{-10} in the most demanding. Tuning δ case by case would tie the cost comparison to a per-case choice, so

the cost experiments use a single conservative target,

$$\delta^\star \;=\; \min_{\text{config, workload}} \delta_{\text{safe}} \;\approx\; 10^{-10}, \tag{5.1}$$

the tightest accuracy any tested case demands. Every configuration is compressed to $\delta^\star$, so each reported cost belongs to one controller that is admissible across every case at once. This also makes the storage figures of section 5.7 conservative. A case whose own δ_{safe} is loose is still held to the tighter $\delta^\star$, and so is compressed less than it strictly needs.

How the variation across runs is reported depends on the metric. Storage (section 5.7) and construction cost (section 5.8) are properties of the operators at the target $\delta^\star$, not of any closed-loop run, so they do not depend on the seed or the workload. Admissibility (section 5.6) and online latency (section 5.9) instead depend on the path the solver takes through the active set, so they vary with workload and seed, and are reported as a range across five seeds and three workloads of each configuration.

5.2. Benchmark system

The benchmark is a mass–spring–damper chain (figure 5.2). It is a line of point masses, each coupled to its neighbours by a linear spring and damper, with every second mass actuated. The chain is a standard benchmark for linear predictive control [44]. It was chosen here for three reasons. First, its dynamics are linear time-invariant (LTI), so the matrix-form reference is unambiguous. Second, it is well understood, so any anomaly in the results points to the tensor format rather than to the plant. Third, its size scales along two independent axes without changing its dynamics, which separates the two size-scaling mechanisms.

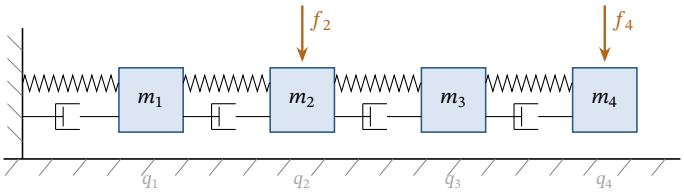


Figure 5.2: The mass–spring–damper benchmark: a chain of k_m masses coupled by springs and dampers, a subset actuated by input forces. The chain length k_m and the prediction horizon N are the two independent scaling axes.

Each mass carries a position and a velocity, so a chain of k_m masses has state $x = [q_1, \dot{q}_1, \dots, q_{k_m}, \dot{q}_{k_m}]^\top \in \mathbb{R}^n$ with $n = 2k_m$. With unit masses, spring constant κ, and damping c, an interior mass obeys $\ddot{q}_i = \kappa(q_{i-1} - 2q_i + q_{i+1}) + c(\dot{q}_{i-1} - 2\dot{q}_i + \dot{q}_{i+1}) + f_i$, where f_i is the applied force when mass i is actuated. Treating each velocity $\dot{q}_i$ as a separate state and writing the per-mass equations as the rows of a single first-order system gives a continuous-time linear model of size n. A zero-order hold at a fixed sampling time discretises it into the LTI form fixed in chapter 3,

$$x_{t+1} = \boldsymbol{A}x_t + \boldsymbol{B}u_t, \qquad \boldsymbol{A} \in \mathbb{R}^{n\times n},\ \boldsymbol{B} \in \mathbb{R}^{n\times m}, \tag{5.2}$$

with the input $u \in \mathbb{R}^m$ collecting the forces on the m actuated masses. The controller minimises the finite-horizon quadratic cost

$$J = \sum_{t=0}^{N-1} \left(x_t^\top \boldsymbol{Q} x_t + u_t^\top \boldsymbol{R} u_t\right) + x_N^\top \boldsymbol{P} x_N, \tag{5.3}$$

with $\boldsymbol{Q} \succeq 0$, $\boldsymbol{R} \succ 0$, and terminal weight $\boldsymbol{P}$ the solution of the discrete algebraic Riccati equation, subject to box bounds on every state and input,

$$x_{\min} \le x_t \le x_{\max}, \qquad u_{\min} \le u_t \le u_{\max}, \tag{5.4}$$

which impose $p = 2(n + m)$ inequality constraints at each step. The chain length k_m therefore sets n, m, and p, while the horizon N sets the number of stages. The next subsection uses this split to define the two scaling axes.

5.2.1. Two scaling axes

There are two scaling axes. The chain length k_m sets the per-stage block sizes n, m, p, the physical dimensions of the plant. The horizon N sets how many steps the controller predicts over.

The spatial sweep grows the chain at a fixed horizon $N = 8$, which enlarges the physical dimensions n, m, p. The horizon sweep fixes a medium chain ($k_m = 40$) and grows N. In both, the dual dimension Np grows, but for different reasons.

Table 5.1: The two size-scaling sweeps. Each varies one axis and holds the other fixed; ranges are smallest to largest.

	Spatial sweep	**Horizon sweep**
Chain length k_m	15–250	40 (fixed)
Horizon N	8 (fixed)	8–81
State dimension n	30–500	80 (fixed)
Input dimension m	8–125	20 (fixed)
Constraints per step p	76–1250	200 (fixed)
Dual dimension Np	608–10,000	1600–16,200

The method is built for large-scale LTI MPC, where the condensed operators become big enough that storing and applying them in matrix form is costly. The sweeps reach that regime but start well below it, so the cost can be tracked as the problem grows. Their upper ends are demanding. The dual dimension Np rises to 10,000 in the spatial sweep and 16,200 in the horizon sweep. The horizon values are picked to factorise into equal factors, which lets the shape plan split each one into constant-size temporal modes (section 4.2). The spatial sweep then runs at a fixed horizon $N = 8$, a standard short-to-moderate prediction window, so temporal growth does not contaminate the spatial trend.

The full configuration table is in appendix B. The code, data, and scripts needed to reproduce every figure and table in this chapter are available at `https://github.com/smaljers01/TT-SEMPC`.

5.2.2. Workloads and reference

The tensor evaluator was predicted to slow down relative to the matrix form as the active set grows, and a constraint was predicted to be violated first near the boundary (section 4.10). Testing either one means running the controller across a range of active-set sizes, not at a single level of constraint activity.

For each seed s, a random direction is drawn uniformly by sampling $\xi_s \sim \mathcal{N}(0, I)$ and normalising it. The initial state is placed along that direction at a fixed fraction α of the distance from the origin to the nearest constraint boundary,

$$x_0(s, \alpha) = \alpha\, r(\xi_s)\, \frac{\xi_s}{\|\xi_s\|}, \qquad \xi_s \sim \mathcal{N}(0, I), \tag{5.5}$$

where $r(\xi_s)$ is the distance to the boundary along ξ_s. Thus $\alpha = 0$ is the origin and $\alpha = 1$ is the boundary itself.

Three workload classes span the range. *Nominal* ($\alpha = 0.25$) starts well inside the feasible set and rarely activates a constraint. *Moderate* ($\alpha = 0.6$) sits in between. *Stress* ($\alpha = 0.8$) starts close to the boundary and drives a large active set. Five seeds are drawn per class, and each case runs for eight receding-horizon closed-loop steps.

The reference is the conventional online QP, solved with operator-splitting quadratic-program solver (OSQP) [18], a widely used operator-splitting solver, on the same condensed problem the SE controllers solve. It is the independent comparison in E0 and E5.

5.3. Validation protocols

The controller is checked in two phases that recur in every experiment. The offline phase builds the operators and is checked for internal consistency. The online phase runs the controller in closed loop and is checked against the matrix-form reference.

5.3.1. Offline validation

The offline phase produces the operators $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}, \bar{\boldsymbol{w}}$, in matrix form for the reference and in TT form for the compressed controllers. Either way, their correctness is checked by the same consistency residuals, built from the defining identities of section 3.3. The inverse-action factors must satisfy $\boldsymbol{HL} = \boldsymbol{F}$ and $\boldsymbol{HM} = \boldsymbol{G}^\top$, and the assembled operators must satisfy $\boldsymbol{S} = \boldsymbol{GM}$ and $\boldsymbol{T} = \boldsymbol{E} + \boldsymbol{GL}$, giving the four relative residuals

$$\begin{aligned} \eta_{\boldsymbol{L}} &= \frac{\|\boldsymbol{HL} - \boldsymbol{F}\|_F}{\|\boldsymbol{F}\|_F}, & \eta_{\boldsymbol{M}} &= \frac{\|\boldsymbol{HM} - \boldsymbol{G}^\top\|_F}{\|\boldsymbol{G}^\top\|_F}, \\ \eta_{\boldsymbol{S}} &= \frac{\|\boldsymbol{S} - \boldsymbol{GM}\|_F}{\|\boldsymbol{GM}\|_F}, & \eta_{\boldsymbol{T}} &= \frac{\|\boldsymbol{T} - \boldsymbol{E} - \boldsymbol{GL}\|_F}{\|\boldsymbol{E} + \boldsymbol{GL}\|_F} \end{aligned} \tag{5.6}$$

each measuring whether the solves and products were computed to the accuracy claimed.

The level they should reach depends on the construction. The matrix-form reference uses no compression, so all four should sit at the numerical precision floor, and it is accepted only when they do. The tensor construction is built to a tolerance δ, so the same residuals should instead sit at order δ.

5.3.2. Online validation

The online phase evaluates the controller in closed loop for eight steps. Eight steps is long enough that the state moves through the constrained region and the active set changes along the trajectory, which exercises the insertion and removal logic of section 3.4. It is short enough that the two controllers' trajectories cannot drift far apart, so any difference between them stays attributable to per-step behaviour rather than to compounding divergence. The acceptance of a compressed controller is judged against the matrix-form controller on the same trajectory by four hard metrics, defined and motivated in section 5.6. Latency is measured by repeated timing. Each online step is first run a few times without recording, so that the data it needs is already cached and any one-off start-up costs are paid. The step is then timed over many runs, and the mean, the 95th percentile, and the maximum are reported.

One guard is applied throughout. A compressed controller is excluded from a latency comparison when its total active-set work differs from the matrix-form controller's by more than a quarter. Writing N^c and N^d for the total active-set iterations of the compressed and matrix-form controllers, this guard is $\Delta_N = |N^c - N^d|/N^d$, and the case is dropped when $\Delta_N > 0.25$. The active-set work is what sets the online cost. It fixes how many solver iterations run and how large the block solves are, and each iteration pays the cost of contracting the operators. A fair timing comparison therefore needs the two controllers to do the same work, which means following the same active-set trajectory. A large divergence in the counts signals the opposite. The compression has changed which constraints the controller activates, the same boundary-flipping effect that produces the constraint violation (section 5.6). The compressed controller is then solving a materially different problem from the reference, so any latency gap reflects the difficulty of that problem rather than the tensor format. Those cases are dropped.

5.4. E0, Validation of the matrix-form reference

Goal. Confirm that the matrix-form SE construction and active-set evaluator of chapter 3 solve the same condensed QP as a conventional online solver. This lets the matrix-form SE controller stand as the reference against which every later compressed controller is judged.

Setup. On each configuration, workload, and seed, the matrix-form controller and OSQP solve the same condensed program,

$$\boldsymbol{U}^\star(x_0) = \arg\min_{\boldsymbol{U}} \ \tfrac{1}{2}\boldsymbol{U}^\top \boldsymbol{H}\boldsymbol{U} + x_0^\top \boldsymbol{F}^\top \boldsymbol{U} \quad \text{s.t.} \quad \boldsymbol{G}\boldsymbol{U} \le \bar{\boldsymbol{w}} + \boldsymbol{E}x_0, \tag{5.7}$$

and are compared on three groups of metrics. All metrics are dimensionless, either relative errors or residuals of normalised quantities, so no units appear here or in the figures.

The first is *agreement* between the two solutions. With $\boldsymbol{U}_{\text{se}}$ and $\boldsymbol{U}_{\text{qp}}$ the two input sequences and $J(\cdot)$ the cost, these are the relative input and objective errors,

$$e_U = \frac{\|\boldsymbol{U}_{\text{se}} - \boldsymbol{U}_{\text{qp}}\|_2}{\|\boldsymbol{U}_{\text{qp}}\|_2}, \qquad e_J = \frac{|J(\boldsymbol{U}_{\text{se}}) - J(\boldsymbol{U}_{\text{qp}})|}{|J(\boldsymbol{U}_{\text{qp}})|}. \tag{5.8}$$

The second is *optimality*, through the four KKT residuals at the multiplier $\boldsymbol{\lambda}$,

$$\begin{aligned} \eta_s &= \frac{\|\boldsymbol{H}\boldsymbol{U} + \boldsymbol{F}x + \boldsymbol{G}^\top\boldsymbol{\lambda}\|_2}{1 + \|\boldsymbol{F}x\|_2}, & \eta_p &= \left\|[\boldsymbol{G}\boldsymbol{U} - \boldsymbol{w}(x)]_+\right\|_\infty, \\ \eta_d &= \left\|[-\boldsymbol{\lambda}]_+\right\|_\infty, & \eta_c &= \left|\boldsymbol{\lambda}^\top(\boldsymbol{G}\boldsymbol{U} - \boldsymbol{w}(x))\right|, \end{aligned} \tag{5.9}$$

which measure stationarity, primal feasibility, dual feasibility, and complementarity.

The third is *internal consistency*, through the LMST residuals $\eta_{\boldsymbol{L}}, \eta_{\boldsymbol{M}}, \eta_{\boldsymbol{S}}, \eta_{\boldsymbol{T}}$ of equation (5.6) and the worst closed-loop constraint violation,

$$v_{\max} = \max_t \left\|[\boldsymbol{G}\boldsymbol{U}_t - \boldsymbol{w}(x_t)]_+\right\|_\infty. \tag{5.10}$$

A fourth quantity, the constraint-activity gap, compares how closely the two solutions approach the active constraints. The metrics above compare the inputs, the cost, and the optimality conditions, but none of them shows whether the two solvers reach the constraint boundary in the same way. This is the behaviour the later experiments depend on. Compression can change which constraints the controller activates (section 5.6), and that change can only be charged to the tensor format if the reference and OSQP agree at the boundary to begin with. The constraint-activity gap records that agreement.

Each normalised inequality $(\boldsymbol{G}\boldsymbol{U} - \boldsymbol{E}x)/\bar{\boldsymbol{w}}$ equals one when its constraint is active. Let a_{se} and a_{qp} be the largest value any inequality reaches over the run, for the matrix-form and OSQP solutions. The constraint-activity gap is their difference,

$$\Delta a = |a_{\text{se}} - a_{\text{qp}}|, \tag{5.11}$$

and, like e_U and e_J, it is recorded as a diagnostic rather than an acceptance criterion.

A case is accepted on three checks. The matrix-form pipeline uses no compression, so its consistency residuals should sit at the double-precision floor. They must stay below 10^{-8}, a generous margin any correct build clears. The worst closed-loop violation must stay below 10^{-4}. The constraints are normalised, so anything smaller lies inside the solver's own feasibility tolerance and is not a real breach. The active-set loop must converge within its iteration cap at the solver's 10^{-9} tolerance. The agreement errors e_U and e_J are recorded as diagnostics.

Result. The matrix-form controller reproduces the conventional solution to solver precision in every case (figure 5.3). The objective error is $e_J \approx 10^{-15}$. The consistency residuals for $\boldsymbol{L}$ and $\boldsymbol{M}$ sit at $\sim 5 \times 10^{-14}$, the rounding level their linear solves leave, the worst closed-loop violation at $v_{\max} \approx 4 \times 10^{-14}$, and the optimality residuals at or below $\sim 7 \times 10^{-13}$, all at the double-precision floor. The residuals $\eta_{\boldsymbol{S}}$, $\eta_{\boldsymbol{T}}$, and η_d are exactly zero by construction, since the reference assembles $\boldsymbol{S}$ and $\boldsymbol{T}$ directly from their defining identities with no solve involved, and the active-set method enforces $\boldsymbol{\lambda} \geq 0$ exactly. Two quantities stand above the floor. The input-sequence error is $e_U \approx 3.7 \times 10^{-7}$, more than eight orders of magnitude above e_J, and the constraint-activity gap $\Delta a \approx 1.5 \times 10^{-11}$ (equation (5.11)).

Analysis. The consistency residuals at the double-precision floor confirm that the inverse-action solves and the operator products of section 3.3 are computed to machine precision. Any deviation that appears once compression is applied in the later experiments is therefore down to the tensor format, not the reference.

The two larger quantities, e_U and Δa, are conditioning artefacts, not evidence of two distinct solutions. The condensed Hessian is ill-conditioned, so along its weakly curved directions the objective is nearly flat near the optimum. Both solvers stop once their residuals fall below tolerance, and the set of inputs meeting that tolerance stretches along those flat directions. The two solvers therefore land 3.7×10^{-7} apart in input while their objective values agree to 10^{-15} and every optimality and consistency residual stays at numerical zero. Because $\boldsymbol{H} \succ 0$ the minimiser is still unique (Assumption 3.1); the gap is in how each solver resolves a near-flat objective, not in the solution. The same reasoning covers Δa. It is computed from the input sequences,

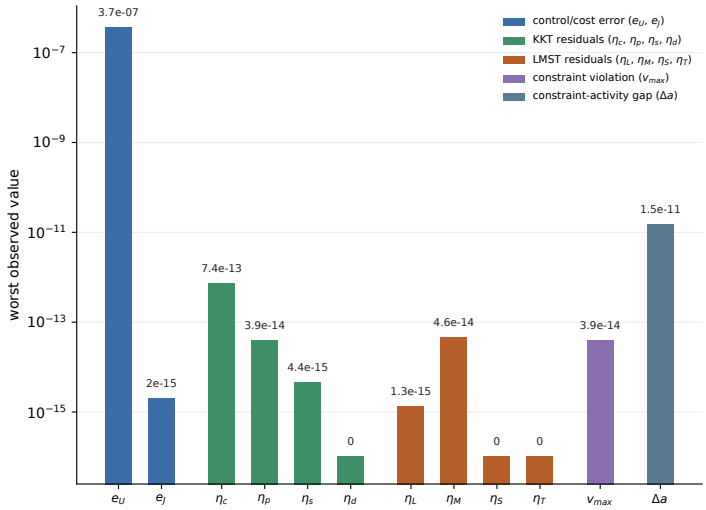


Figure 5.3: E0: worst observed value of each metric across all runs (log scale). The objective error, consistency residuals, constraint violation, and optimality residuals lie at the double-precision floor. The input-sequence error e_U and the constraint-activity gap Δa reflect the near-flat directions of the ill-conditioned objective.

so the spread of near-optimal inputs along the flat directions carries into it, leaving it at $\sim 10^{-11}$ rather than at the floor. Acceptance therefore rests on the objective and the residuals, with e_U and Δa kept only as diagnostics.

Scope. E0 fixes a clean baseline, so that every deviation in the later experiments can be charged to the tensor format rather than to the construction pipeline or the solver. The residual floors are numerical, set by double precision on this hardware, so they also fix the resolution of the later comparisons. A compression error below the $\sim 10^{-13}$ floor reported here cannot be separated from round-off. Agreement is measured against a single conventional solver, but the optimality residuals of equation (5.9) certify the matrix-form solution intrinsically, so the reference stands independently of that solver. It is read from the objective and the residuals, not from the input sequence, whose conditioning-limited error e_U is kept only as a diagnostic, as is the constraint-activity gap Δa. The same reasoning applies wherever input-level differences are interpreted later.

5.5. E1, Target tracking and proxy validity

Goal. E1 establishes two properties: that the requested final-operator target δ produces a perturbation of $(\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T})$ that is predictable and independent of the construction route, and that the late-stage route is a valid proxy for the final-operator error of the tensorised construction routes.

Route dependence. The three construction routes of chapter 4 reach the same operators, but they apply TT rounding at different points. They are therefore not guaranteed to reach the same achieved error at a given request δ.

The late route forms the matrix-form operators exactly and rounds once, at the end. Its error is a single truncation, which the bound of section 2.3 ties directly to δ. The mid route condenses in the tensor format using the Kronecker-sum structure of section 4.4 and then solves the inverse action. Rounding is applied repeatedly through assembly, and the iterative solve introduces its own residual. The early route condenses natively from the system primitives. It is the longest chain of rounded operations and accumulates the most error.

Repeated rounding may compound, and the iterative solves of the mid and early routes cannot reduce their residual below their own tolerance. A request for δ could therefore yield a larger achieved error, or one that depends on the route. E1 determines whether this occurs. The outcome decides whether δ is a single shared control parameter or three distinct ones. Were the routes to diverge, the late route could not stand in for the others, and E2 and E3 would have to be repeated for each route.

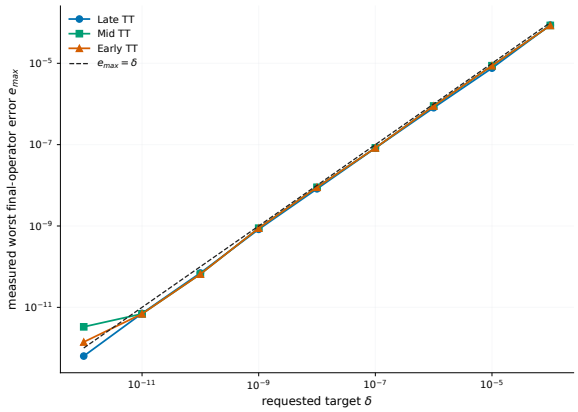
Setup. On the medium configuration, the operators are compressed at each $\delta \in \{10^{-4}, 10^{-5}, \dots, 10^{-12}\}$ by the late, mid, and early routes. Three quantities are measured.

The first is the achieved error, defined as the worst relative operator error over the four operators,

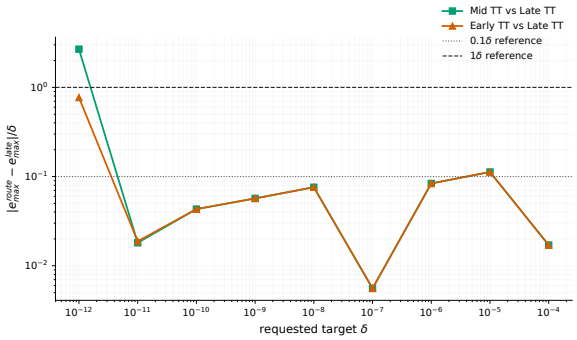
$$e^{\max} = \max_{\boldsymbol{O} \in \{\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}\}} \frac{\|\widetilde{\mathcal{O}} - \boldsymbol{O}\|_F}{\|\boldsymbol{O}\|_F}, \tag{5.12}$$

with $\widetilde{\mathcal{O}}$ the compressed operator. The target-tracking ratio $\eta = e^{\max}/\delta$ converts this into a pass/fail test of whether the request is met, and is accepted when $\eta \leq 1$ over the usable range. The second is the route gap, $|e^{\max}_{\text{route}} - e^{\max}_{\text{late}}|/\delta$, which tests proxy validity and is accepted when it remains below 0.1. The third is the maximum TT rank, which measures the cost of accuracy. The configuration is held fixed, so the only variables are the target and the route.

Result. Across the range 10^{-4} to 10^{-11}, all three routes track the requested target. The measured worst error $e^{\max}$ stays within about 10% of δ and, in every case, below it, so each passes the $\eta \leq 1$ criterion (figure 5.4a). The mid and early routes differ from the late route by between 1% and 10% over this range (figure 5.4b). The late route is therefore a valid proxy for the other two.



(a) Worst final-operator error versus target. The three routes coincide on the $e^{\max} = \delta$ line, so the target δ is delivered route-independently.



(b) The route-to-late error gap, normalised by δ, stays between 1% and 10% down to 10^{-11}, which justifies using the late route as the proxy for final-operator perturbations in this accuracy range.

Figure 5.4: E1 results. (a) Worst final-operator error versus target. (b) Route-to-late error gap normalised by the target tolerance.

This accuracy comes at the cost of rank. As the target tightens, the maximum rank rises from about 120 to 790 (figure 5.5). The three routes reach almost the same rank at each target. The late route stays a few percent below the mid and early routes, which themselves nearly coincide. Plotting the rank against $\log(1/\delta)$ gives a straight line, with coefficient of determination $R^2 = 0.99$, so the rank grows in proportion to $\log(1/\delta)$.

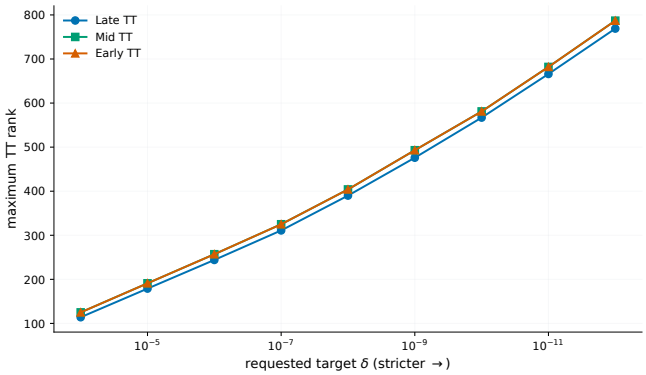


Figure 5.5: E1: maximum TT rank versus target. Accuracy is bought with rank: the maximum rank grows as the target tightens, and the three routes nearly coincide.

Below 10^{-11} the mid- and early-route errors stop improving and settle at a floor, while the late route keeps improving (figure 5.4a).

Analysis. The linear relation $e^{\max} \propto \delta$ confirms that the tolerance schedule of section 4.9 keeps the accumulated rounding error of each stage just below the target.

The rank growth matches the prediction of section 4.10. On a stable plant the operators decay geometrically, meaning that blocks of the operator far from the diagonal shrink by a constant factor at each step. To reach accuracy δ only the blocks larger than δ must be kept, and their number grows like $\log(1/\delta)$. Each kept block adds a fixed amount of rank, so the rank grows like $\log(1/\delta)$, in agreement with the straight-line fit above.

The floor below 10^{-11} is set by the solver residual budget of section 4.7. It originates in the iterative solve, not in the tensor representation. The late route uses no iterative solve and therefore carries no such residual, so its own floor falls below the smallest target tested here.

The per-operator breakdown of figure 5.6 shows that $\boldsymbol{S}$ and $\boldsymbol{M}$ account for most of the error. The same two operators dominate the storage cost in E3.

Scope. The proxy holds over a bounded accuracy range. The late route matches the mid and early routes in both error and rank from 10^{-4} down to 10^{-11}, but below 10^{-11} the iterative routes settle at their residual floor and the match breaks. Any later result that leans on the late route is therefore limited to targets at or above 10^{-11}. The match is also established at a single problem size, the medium configuration, so it bears on the choice of construction route, not on how the operators scale with the system.

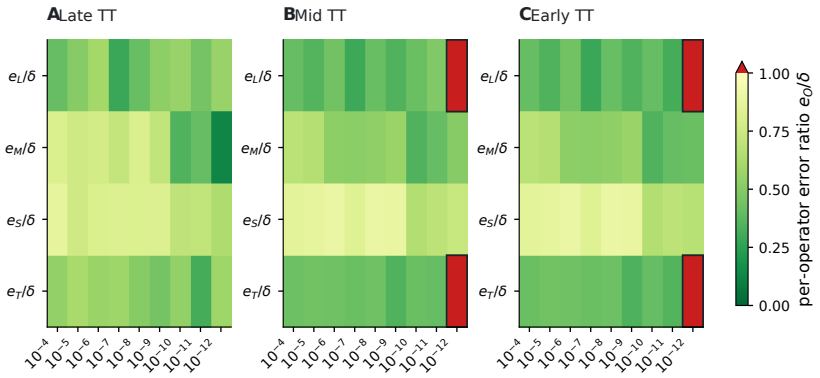


Figure 5.6: E1: per-operator error decomposition. The dual-coupling matrix $\boldsymbol{S}$ and the inverse-action factor $\boldsymbol{M}$ carry most of the approximation error.

5.6. E2, Closed-loop admissibility

Goal. E2 answers three questions: which final-operator perturbation magnitudes keep the compressed controller admissible in closed loop, how the admissible target scales with the two size axes, and which mechanism causes the failure when it occurs.

Setup. Each of the seven configurations is run under the three workloads described in section 5.2.2. The late compressed controller is run across the δ grid and compared against the matrix-form controller on the same trajectory. Four metrics summarise the comparison.

Constraint satisfaction is measured by the worst closed-loop constraint violation $\upsilon_{\max}$ of equation (5.10). It is the largest amount by which any constraint of equation (3.8) is breached along the run. The input and state deviations measure how far the compressed trajectory drifts from the matrix-form one,

$$e_u^{\mathrm{cl}} = \frac{\|U^c - U^d\|_2}{\|U^d\|_2}, \qquad e_x^{\mathrm{cl}} = \frac{\|X^c - X^d\|_2}{\|X^d\|_2}, \tag{5.13}$$

with U, X the stacked closed-loop inputs and states, and the superscripts c, d the compressed and matrix-form controllers. The cost change $\Delta J = |J^c - J^d|/|J^d|$ is the relative loss of closed-loop performance, with J the accumulated stage cost of equation (5.3) over the run.

The thresholds follow the accuracy required of a constrained predictive controller, with constraint satisfaction as the primary criterion. The constraints encode the physical and safety limits the controller exists to enforce, so a controller that violates them is inadmissible regardless of how cheap or accurate it is [33]. A run is therefore rejected unless $v_{\max} \leq 10^{-4}$, the same threshold as in section 5.4 and for the same reason, since the constraints are normalised and a smaller margin lies inside the solver's own feasibility tolerance.

The trajectory and cost metrics are secondary checks on a controller that already satisfies its constraints. The input and state deviations must stay below 5%. This is finer than the model uncertainty of any plant for which a nominal linear model would be used, so a smaller drift is already masked by modelling error. The deviation here is unlike the conditioning artefact of section 5.4. There, two solvers solved the same problem, so any input gap was numerical noise. Here, the compressed controller solves a perturbed problem, so a nonzero deviation is a real effect of compression. The cost change must stay below 1%. This is negligible next to the suboptimality introduced by the finite horizon and the receding-horizon approximation [32].

A run passes when all four metrics clear their thresholds and the active-set loop converges at every step. The admissible target δ_{safe} is the loosest δ that passes, provided every tighter δ passes as well. The condition on the tighter values rules out a single spurious pass at a loose tolerance. The full per-metric tables are in appendix C.4.

Result. Acceptance depends on the workload. All 315 nominal cases (seven configurations, five seeds, nine targets) pass, 281 of 315 moderate cases pass, and 243 of 315 stress cases pass (figure 5.7). The controller remains admissible when few constraints are active, and admissibility becomes harder to maintain as the active set grows. Because every nominal case passes at the loosest tested target, the nominal δ_{safe} is right-censored at 10^{-4} and could be looser.

Figure 5.7: E2: admissible target δ_{safe} across configuration × workload. Admissibility is workload-driven, tightening as the active set grows.

The scaling of the admissible target is the central finding (figure 5.8). The clearest dependence is on the horizon under the stress workload, where δ_{safe} tightens from 10^{-9} at $N = 8$ to 10^{-10} at the longest horizons, though not strictly monotonically. Spatial growth shows no clear trend. Most spatial cases stay between 10^{-5} and 10^{-7}, so system size has no consistent effect on the admissible target.

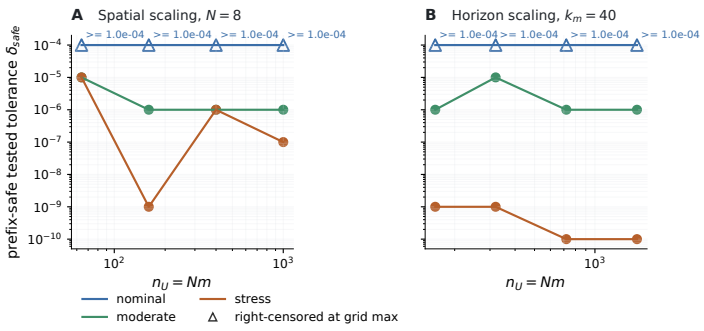


Figure 5.8: E2: admissible target δ_{safe} versus problem size $n_U = Nm$ under spatial growth (left) and horizon growth (right). Horizon growth tightens the target far more than spatial growth.

The two sweeps share the medium configuration at $N = 8$, whose stress target is 10^{-9}. This is the loosest point of the horizon sweep. Among the spatial cases it is far tighter than its neighbours, which sit at 10^{-5} to 10^{-7}. The same point is therefore both the single spatial outlier and the cause of the shallow start of the horizon trend. If it instead matched its spatial neighbours, the horizon tightening would span three to four orders of magnitude rather than one, making the trend predicted in section 4.10 clearer still.

In every transition from accepted to rejected, the constraint-violation metric $v_{\max}$ crosses its limit first, ahead of the input and state metrics (figure 5.9).

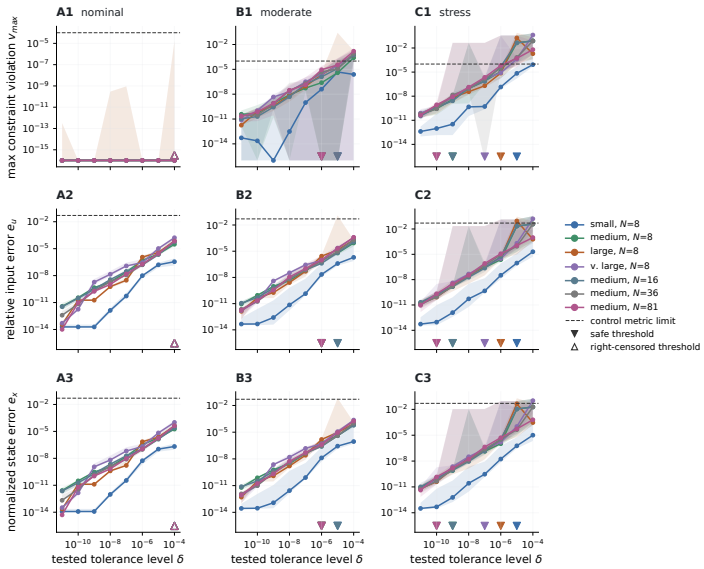


Figure 5.9: E2: closed-loop metric bands against δ for all seven configurations. The constraint-violation metric $v_{\max}$ crosses its threshold before the input and state metrics in every accepted-to-rejected transition. Open markers show right-censored cases that pass at every tested target.

Analysis. The online loop of section 3.4 uses the compressed operators at three points, so the compression error enters at each. It first forms the starting residual $\boldsymbol{v} = -(\bar{\boldsymbol{w}} + \boldsymbol{T}x_0)$ from $\bar{\boldsymbol{w}}$ and $\boldsymbol{T}$ (equation (3.14)). It then solves the reduced dual system $\boldsymbol{S}_{\mathcal{A}\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{v}_{\mathcal{A}}$ for the multipliers through $\boldsymbol{S}$ (equation (3.15)). It finally reconstructs the input $\boldsymbol{U}^\star = -\boldsymbol{L}x_0 - \boldsymbol{M}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$ through $\boldsymbol{M}$ (the state-driven baseline $-\boldsymbol{L}x_0$ also carries an error, but its effect on the constraints is already represented by $\boldsymbol{T} = \boldsymbol{E} + \boldsymbol{G}\boldsymbol{L}$). Each operator carries a relative error of order δ, which passes into the residual, the multipliers, and the reconstructed input in turn.

The active-set update is where this error changes the outcome. At each step the loop adds the most violated inactive constraint and drops any constraint whose multiplier has turned negative (Algorithm 3). Both decisions are sign comparisons, on the residual $\boldsymbol{v}$, on the multipliers, and on the updated residual $\boldsymbol{r} = \boldsymbol{v} - \boldsymbol{S}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$. Near a constraint boundary one of these quantities is close to zero, so an $\mathcal{O}(\delta)$ error can flip its sign. The loop then adds or drops a different constraint, and the controller commits to a different input that may violate a different constraint.

The worsening with horizon confirms the prediction of section 4.10 that the conditioning of $\boldsymbol{S}$ grows worse under condensing as the horizon increases. A longer horizon couples more constraints through $\boldsymbol{S}$ and $\boldsymbol{M}$, so a single perturbation can disturb more of them at once. Figure 5.10 separates this sensitivity from raw workload. The active-set work grows along the spatial sweep, where the per-step constraint count grows, and stays roughly flat along the horizon sweep. The spatial cases therefore do more work yet keep a loose target, while the horizon cases tighten at constant work. What tightens δ_{safe} is the reach of a perturbation through the coupled operators, not the amount of active-set work the loop performs. Spatial growth at fixed horizon enlarges the per-step blocks but not the temporal coupling. The reach of a perturbation therefore stays roughly fixed, and δ_{safe} does not tighten monotonically.

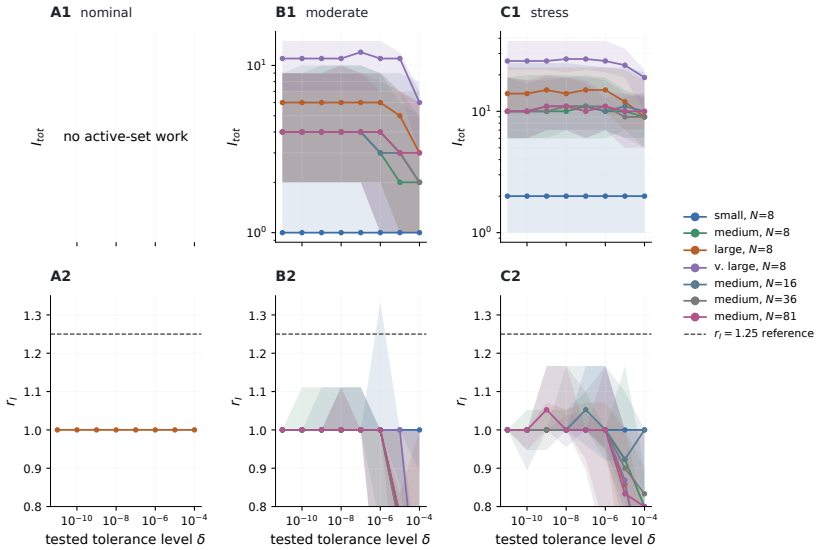


Figure 5.10: E2: active-set work by workload and configuration. Nominal cases do almost none. The work grows with the per-step constraint count along the spatial sweep and stays roughly flat along the horizon sweep, so the horizon tightening of figure 5.8 is not explained by extra work.

That the constraint violation crosses first confirms the prediction of section 4.10 that constraint satisfaction is lost before optimality, and it follows from the first-order-versus-second-order split predicted there. A small operator error can only flip an active-set decision at a constraint that is marginally active, where the sign test is near zero and the multiplier is almost zero, so the input it produces differs from the exact input by an amount of order δ. The achieved cost responds to that difference only at second order. The cost is quadratic and the exact input nearly minimises it, since the flipped constraint was barely binding, so an $\mathcal{O}(\delta)$ input error raises the cost by $\mathcal{O}(\delta^2)$. The constraint violation responds at first order, because it is linear in the input. An $\mathcal{O}(\delta)$ input error places the input $\mathcal{O}(\delta)$ past the boundary, which the hard $v_{\max}$ threshold registers. The same small input error therefore breaches a constraint while leaving the cost almost unchanged. The input and state deviations of equation (5.13) average over the whole run and move at only a few steps, so they too stay small. The controller violates a constraint while still tracking the matrix-form trajectory closely and staying near its cost. All four thresholds apply, but constraint satisfaction is the one that binds in every tested transition, so it is what sets the admissible target in practice.

The same mechanism explains why the Frobenius operator error of E1 does not predict admissibility. The error is an average of the operator perturbation. It does not capture which entries were perturbed, or how close the trajectory runs to a constraint boundary, and these are what decide admissibility, through the active-set sensitivity at the boundary rather than the average operator accuracy. A nominal case, whose trajectory stays far from its constraints, can therefore tolerate 10^{-4}, while its stress counterpart, which runs close to them, needs 10^{-10}.

Scope. The eight-step window and five seeds set the resolution of this result. δ_{safe} is tested at order-of-magnitude spacing from a short trajectory, so its value is pinned only to within a grid step, and the magnitude of the horizon trend in particular hinges on the single medium-configuration point flagged above. The acceptance rule is empirical. It certifies admissibility on the tested trajectories and is not a recursive-feasibility or stability guarantee, so a formal closed-loop analysis under a bounded operator perturbation remains open. And the result is consumed conservatively. The long-horizon stress case, at $\delta_{\text{safe}} \approx 10^{-10}$, fixes the single target $\delta^\star$ of equation (5.1) applied to every case in the rest of the chapter, so the downstream storage and latency figures use a target tighter than most cases require.

These limits point to one direction for future work. The target here is read from a uniform operator error and then fixed at the worst case, yet the analysis above shows admissibility is governed by active-set sensitivity at the constraint boundary, not by average operator accuracy. A targeted variant of E2 that perturbs the operators only under the stress workload, where the trajectory rides the constraint boundary, would isolate this sensitivity directly. A criterion that bounds the closed-loop constraint violation directly would relax the accuracy away from the boundary and tighten it only where it matters, replacing the conservative $\delta^\star$ with a per-configuration target.

5.7. E3, Storage compression

Goal. E3 measures the storage compression of the final operators at the conservative admissible target $\delta^\star$, and identifies which operator and which property of the factorisation set the compression ratio.

Setup. The matrix-form operators are compressed by the late route at the fixed target $\delta^\star$ of equation (5.1). The result is reported as the matrix-to-TT storage ratio

$$\rho_{\text{storage}} = \frac{B_{\text{mat}}}{B_{\text{TT}}}, \qquad B_{\text{TT}} = \sum_k R_{k-1} I_k J_k R_k, \tag{5.14}$$

counted in the number of stored scalar parameters. Here B_{TT} is the storage equation (2.4) and B_{mat} is the matrix-form entry count. Three further quantities are reported: the per-operator compression ratios ρ_O, each operator's share of the total storage $\phi_O = B_O/B_{\text{TT}}$, and the per-operator maximum ranks.

A configuration is storage-beneficial when $\rho_{\text{storage}} > 1$. Every case is compressed to the single conservative $\delta^\star$, which E2 certifies as admissible for all configurations. The reported ratios are therefore lower bounds. A case whose own δ_{safe} is looser would compress further. Storage counts scalar entries with no index overhead.

Result. Every configuration is storage-beneficial, with ρ_{storage} from 1.4× to 36.3× (figure 5.11). The horizon sweep gives the largest and cleanest gains, growing 1.4 → 3.2 → 9.7 → 36.3× as N grows from 8 to 81. The spatial sweep is non-monotone, at 2.9, 1.4, 7.6, 2.8×.

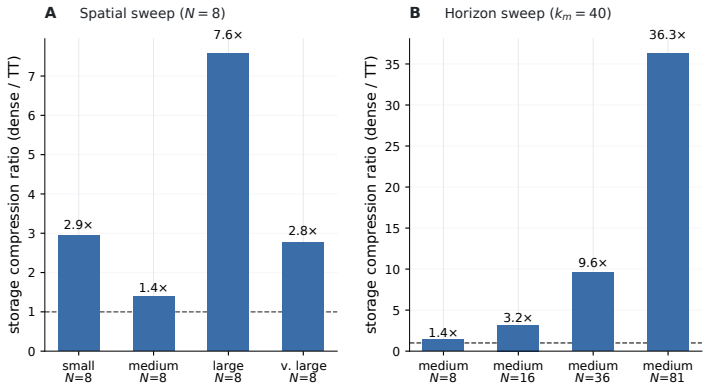


Figure 5.11: E3: storage ratio versus configuration. The gain is clean and monotone under horizon growth, and non-monotone under spatial growth.

The gain comes almost entirely from the dual-coupling matrix $\boldsymbol{S}$. This is the largest of the five operators, with one row and one column per constraint, and it holds between 68% and 85% of the total storage. Together with $\boldsymbol{M}$ it accounts for nearly all of the compression. The operators $\boldsymbol{L}$ and $\boldsymbol{T}$ compress only modestly, and $\bar{\boldsymbol{w}}$ is kept in vector form (figure 5.12). The total ratio is therefore close to the compression of $\boldsymbol{S}$ alone.

The operators also differ widely in TT rank, so no single maximum rank summarises the set. At the largest configuration the dual-coupling matrix $\boldsymbol{S}$ reaches a maximum TT rank of 969, while $\boldsymbol{M}$ reaches 160 and $\boldsymbol{L}$ only 79 (table C.8). A maximum rank taken over the operators therefore reflects $\boldsymbol{S}$ alone, and because the external dimensions change from one configuration to the next, even that figure is not comparable across cases. The storage ratio of equation (5.14), which counts every operator's ranks and mode sizes, is therefore the metric reported throughout.

The matrix-form storage grows quadratically with the constraint count Np, as expected from its largest operator $\boldsymbol{S}$ being $Np \times Np$. A log-log fit confirms this, $B_{\text{mat}} \propto Np^{1.98}$ with $R^2 = 1.00$.

The rank burden $R_{\max}(\boldsymbol{S})/Np$, the maximum TT rank of $\boldsymbol{S}$ divided by its dimension, falls from 0.35 to 0.06 along the horizon sweep. The TT storage grows as $B_{\text{TT}} \propto Np^{0.57}$ with $R^2 = 0.98$, well below the quadratic growth of the matrix form, and the compression ratio grows as $Np^{1.4}$, reaching 36× at the longest horizon (figure 5.13).

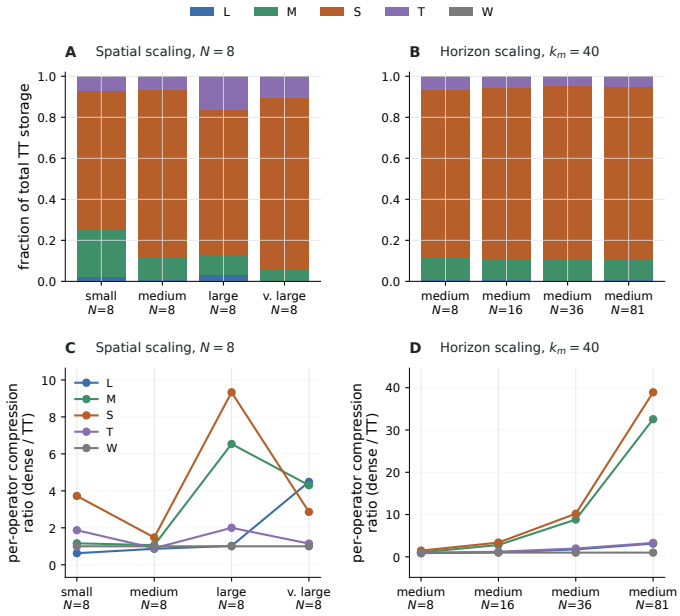


Figure 5.12: E3: per-operator storage share. The dual-coupling matrix $\boldsymbol{S}$ holds most of the total storage, so the total ratio is close to that of $\boldsymbol{S}$ alone.

The spatial sweep exposes a second driver of storage. The large and very-large configurations have near-identical rank burdens, 0.051 and 0.055, yet compress by 7.6× and 2.8×. Something besides the rank must set the storage, and the shape-plan variation below identifies it.

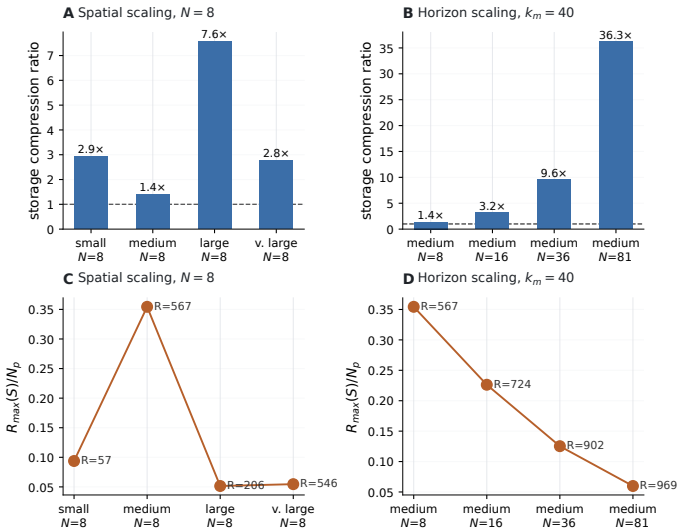


Figure 5.13: E3: total compression against the rank burden $R_{\max}(\boldsymbol{S})/Np$.

Shape-plan variation. To isolate this second driver, the shape plan was varied one dimension at a time (table 5.2). A problem has a single temporal dimension, the horizon N, and several physical dimensions, the per-step state, input, and constraint sizes. The baseline keeps each dimension in a single core. Splitting the temporal dimension into smaller cores keeps the parameter count between 91% and 106% of the baseline. Splitting the physical dimensions reduces it to as little as 13%.

The large configuration shows this directly. Splitting the physical dimension of $\boldsymbol{S}$ from $p = 500$ into 10×10×5 reduces the parameter count from 13.25 million to 1.32 million, while the maximum rank rises from 53 to 166. The storage falls even though the rank rises.

Table 5.2: E3: shape-plan variation. Parameter count and maximum TT rank relative to a single-temporal, single-physical baseline. Splitting the physical dimensions cuts storage sharply while raising the rank, because the per-core mode products fall faster than the rank grows.

Dimensions split	Acts on	Parameter ratio	Rank ratio
Temporal only	horizon dimension	0.91–1.06	1.0–1.07
Physical only	state / input / constraint	0.13–0.70	1.5–8.6
Both	all dimensions	0.13–0.78	1.5–8.8

This is the general pattern, not a feature of one configuration (figure 5.14). The comparison uses two ratios. The parameter-count ratio divides the split plan's parameter count by the single-core baseline's, and the maximum-rank ratio does the same for the maximum TT rank. Across every configuration, splitting the physical dimensions moves the two quantities in opposite directions. It drops the parameter count to between 13% and 78% of the single-core baseline and raises the maximum rank to between 1.5 and 8.8 times it. Splitting the temporal dimension does neither, leaving the parameter count within about 10% of the baseline at almost unchanged rank. Splitting the physical dimensions therefore moves the storage. Splitting the temporal dimension does not.

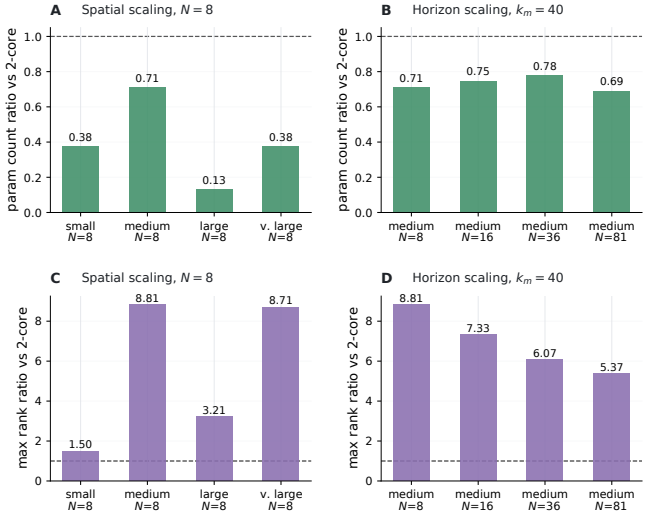


Figure 5.14: E3: parameter count (top) and maximum TT rank (bottom) of a split factorisation relative to the single-core baseline, across the spatial and horizon sweeps. Every split plan falls below one in parameters while rising above one in rank. Storage and rank move in opposite directions, so the rank cannot be what governs storage.

Analysis. The storage of equation (2.4), $B_{\mathrm{TT}} = \sum_k R_{k-1} I_k J_k R_k$, depends on two quantities: the ranks R_k and the mode products $I_k J_k$. Reducing either reduces the storage, and the shape-plan variation above separates the two. Splitting the physical dimensions lowers the mode products, which is why it reduces the storage even as the rank rises. Splitting the temporal dimension changes neither appreciably, because the temporal cores hold only a small share of the storage, which is dominated by the physical p^2 block of $\boldsymbol{S}$, held in the physical core. The horizon and the physical factorisation therefore act on different factors of the same sum: the horizon on the ranks, the physical factorisation on the mode products.

The horizon reduces storage through the rank. The matrix form of $\boldsymbol{S}$ stores all Np^2 entries, so it grows quadratically with the horizon, while the rank of $\boldsymbol{S}$ grows only logarithmically (section 4.10). The TT size therefore grows far more slowly, and the compression ratio rises steadily to 36× at the longest horizon. The horizon values are chosen to factorise into equal factors so that the temporal modes stay small as the horizon grows. This keeps the temporal cores cheap, but the increase in compression comes from the falling rank burden of $\boldsymbol{S}$, not from the factorisation.

The spatial trend follows from the same two factors, but less cleanly. Most of its non-monotonicity is still due to its ranks. The medium configuration, where the two sweeps meet, carries a far higher rank burden than

its neighbours (about 0.35 versus 0.05 to 0.09), so its $\boldsymbol{S}$ is not low-rank in TT form and it compresses the least of the sweep. The part the rank cannot explain is the two largest spatial configurations, which compress by 7.6× and 2.8× at almost equal rank burden. At equal rank only the mode products differ, so there the physical factorisation decides. The large configuration's physical sizes factor into more and smaller modes than the very large one's, and it compresses almost three times as much, so when a system's sizes admit a finer factorisation, more compression is available. Unlike the horizon, the physical sizes n, m, p are fixed by the system rather than chosen, so the mode products a shape plan can reach depend on how those sizes happen to factor. That dependence is irregular and does not improve with size, so the spatial compression varies non-monotonically while the horizon compression rises steadily.

Scope. Three limits bound this result. The first is the accuracy target. Every case is held to the single conservative $\delta^\star$ that E2 fixes from the worst configuration. A case whose own δ_{safe} is looser would compress further, so a per-configuration target would raise the figures. The rank law of section 5.5 suggests by how much. The rank grows with $\log(1/\delta)$, so a nominal case admissible at 10^{-4} but held to $\delta^\star \approx 10^{-10}$ carries roughly two and a half times the rank it needs. Storage grows with the square of the rank, so that case stores roughly six times more than its own target requires. This estimate extrapolates a law measured on the medium configuration, so it is indicative rather than exact. The second is the shape plan. Only one plan per operator was swept, so the non-monotone spatial pattern is explained but not optimised. The physical sizes n, m, p are also fixed by the system, so the compression a shape plan can reach on the spatial axis is bounded by how those sizes factor. Even the storage-optimal plan, found by searching the mode sizes of section 4.2, stays limited by the dimensions the system provides. The third is the operator structure. The horizon compression rests on $\boldsymbol{S}$ being low-rank in TT form, which follows from the geometric decay of the operators on this stable structured family (section 4.10). A system whose $\boldsymbol{S}$ lacks that structure would compress less. So the laws here are not claimed beyond this family, and confirming them on other structured operators remains open.

5.8. E4, Construction cost

Goal. E4 determines whether the tensor-native construction routes reduce offline time or peak memory relative to matrix-form construction at the conservative target, and locates the bottleneck if they do not.

Setup. The matrix-form, late, mid, and early routes are built at the fixed target $\delta^\star$ under a 12 gigabyte memory cap. The comparison records four quantities: the total construction time, the inverse-action solve-phase time, the peak resident memory split into pre-solve, solve, and post-solve phases, and the route status. The status marks whether a route reached the target, exhausted memory, or failed.

A route is construction-beneficial only if it reaches the admissible target within the memory cap and is faster or computationally lighter than the matrix form. A route that cannot reach the target, or that hits the cap, is reported as a failure. The matrix-form route is the baseline. **Result.** No tensor-native route is construction-beneficial in any case. Where the mid and early routes complete, they are one to four orders of magnitude slower than the matrix form, up to about 22,800× on the medium-size long-horizon case. On the largest spatial and longest-horizon cases they fail outright, exhausting memory at exactly the configurations where E3 found the final operator most compressible (figure 5.15).

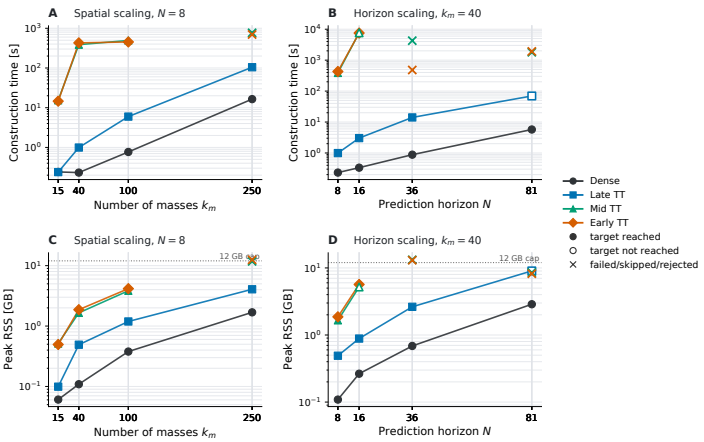


Figure 5.15: E4: construction time and peak memory against size. Matrix-form and late complete throughout; mid and early are orders of magnitude slower where they complete and fail (markers) on the largest configurations.

The cost sits almost entirely in one phase. The inverse-action solve is between 96.1% and 99.9% of construction time across all completed runs (figure 5.16), and it holds the entire peak memory. The initial tensorisation of the condensed operators is at most about a tenth of the run peak (figure 5.17). The construction time tracks the operator rank, rising roughly with its square, and barely changes with the problem size Np.

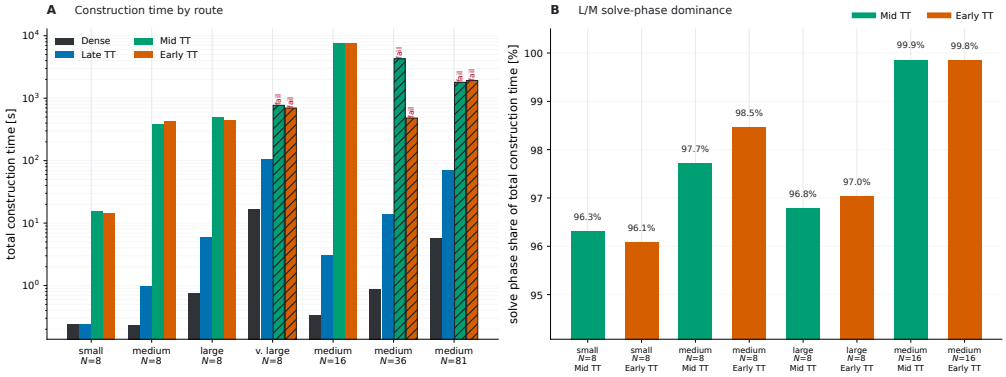


Figure 5.16: E4: solve-phase share of construction time, 96.1%–99.9%. The inverse-action solve dominates construction time.

Analysis. The bottleneck matches the mechanism predicted in section 4.9. The solver reaches a low-rank result only by passing through local subproblems whose ranks run well above the final rounded rank. The peak cost is therefore set by the largest intermediate rank, not by the compact result. A compressible final operator does not imply a cheap construction, because the rounding tolerance bounds the final operators but not the ranks visited on the way to them.

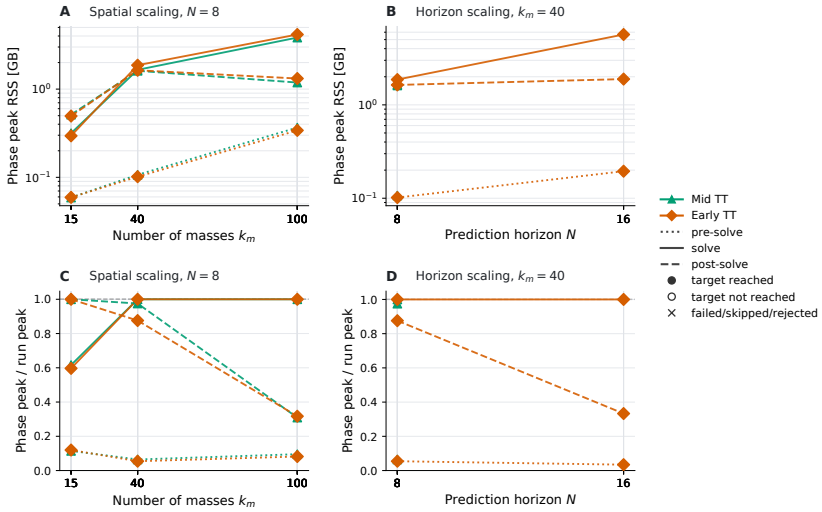


Figure 5.17: E4: peak memory by phase. The peak sits in the solve and assembly phases, not the setup.

The multiple-RHS $\boldsymbol{M}$-solve is the worst case since its right-hand-side count grows with both the horizon and the constraint count. The column-chunked PCG of section 4.7 keeps the transient memory of each per-core solve small, but the run peak is set by the intermediate iterate, held in TT form across every sweep. Chunking cannot reduce that object, so the solve dominates both time and memory. These intermediate ranks are inherent to the inverse action at the required accuracy, so reducing the construction cost requires a formulation that keeps the inverse action low-rank throughout, not only in its result.

The data confirm this. A log-log fit over the completed mid-route runs gives $t \propto R_{\max}(\boldsymbol{S})^{2.0}$ with $R^2 = 0.80$. The construction time is therefore close to quadratic in the rank, consistent with the prediction of section 4.10. The $k_m = 100$ case, large but low-rank at $Np = 4000$ and $R_{\max}(\boldsymbol{S}) = 206$, builds in about 5×10^2 seconds, while the smaller but harder $N = 16$ case at $Np = 3200$ and $R_{\max}(\boldsymbol{S}) = 724$ needs about 8×10^3 seconds, ten times longer at the smaller size. The final rank predicts the blowup even though the intermediate rank is its cause, because a harder operator has both.

At the tested sizes, matrix-form construction builds $\boldsymbol{S}$ quickly and within the memory cap, so construction is not the bottleneck here. The reason to build in the tensor format is to go beyond these sizes. The matrix-form route is capped by having to form the full $Np \times Np$ matrix $\boldsymbol{S}$. Once $\boldsymbol{S}$ no longer fits in memory, that route cannot run. The mid and early routes were designed to avoid this. They build the operators directly in compressed form and never assemble the dense $\boldsymbol{S}$. If they worked, construction cost would track the compressed size rather than the dense matrix size, and the method would scale to problems the matrix-form route cannot reach.

Neither route works in practice. Both run orders of magnitude slower than matrix-form construction, and both exhaust memory on the largest cases. The cause is the inverse-action solve for $\boldsymbol{M}$. Its final result is low-rank, but the solve passes through intermediate TT ranks far larger than that result, and these set the peak memory and time. The only route that reaches the compressed operators is the late route, which forms the dense matrix operators and rounds them at the end. Construction therefore still passes through the full $\boldsymbol{S}$. The tensor format shrinks the controller that is stored and applied (E3, E5), but not the object that must be built to obtain it.

Scope. Three limits bound this result. The first is the budget. The slowdowns and the points of failure are specific to the 12 gigabyte cap and this machine. A larger budget would move where the routes fail, but not the rank-driven scaling behind it. The second is the target. Every route is built at the single conservative $\delta^\star$. The intermediate ranks scale with the required accuracy, so a looser per-configuration target would lower the construction cost and push the failures outward. The third is the solver. One implementation was used, which handles the multiple RHSs in column chunks, so part of the bottleneck's severity is a property of that choice, not of the format. Whether any construction can avoid the high intermediate ranks is therefore still open. The clearest direction is a multiple-RHS inverse action that stays low-rank throughout, for example by exploiting the Kronecker-sum structure of section 4.4 or by solving the constraint columns jointly [25].

5.9. E5, Online latency

Goal. E5 determines whether the matrix-form controller is faster online than the conventional QP, whether the tensor controller is faster than the matrix-form one, and how the storage result of E3 relates to online cost.

Setup. The online latency is timed at the fixed target $\delta^\star$, for each configuration and workload. Three controllers are compared: the conventional QP, the matrix-form controller, and the tensor controller. Each is timed per online step under the repeated-timing protocol of section 5.3.2. The mean, the 95th percentile, and the maximum are reported.

Some records are excluded so that the comparison is between controllers solving the same problem. A record is dropped if it is not E2-admissible, or if its compressed active-set work differs from the matrix-form controller's by more than a quarter. The conventional program is the independent reference. The main comparison is the tensor controller against the matrix-form one.

Result. The matrix-form controller is fastest everywhere, beating the conventional program by one to three orders of magnitude in mean latency and by two to three in the maximum latency (figure 5.18). The tensor controller sits between the two. It beats the conventional program but is 2 to 54× slower than the matrix-form controller, and never faster. These are median slowdowns over the fifteen workload-seed runs of each configuration. The per-run spread is wide and itself workload-driven, from an interquartile range of $[2.2, 3.2]\times$ at the smallest configuration to $[19, 172]\times$ at the largest, because the stress workloads that drive the largest active sets also drive the largest penalties.

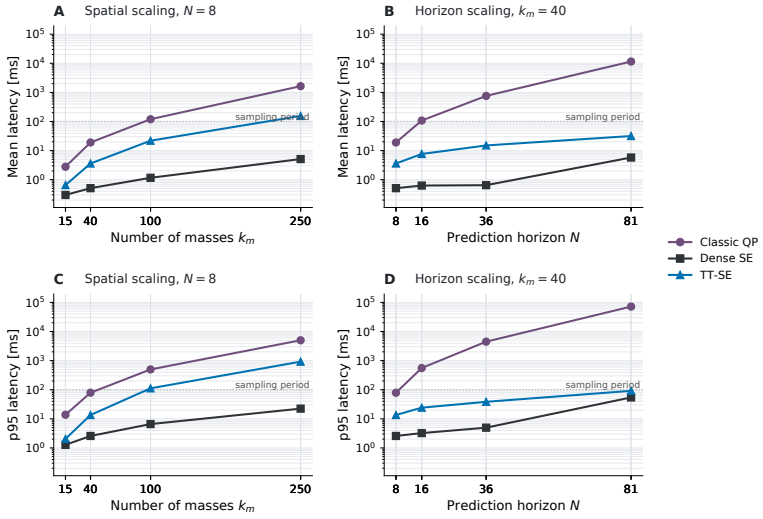


Figure 5.18: E5: per-step online latency against size. The matrix-form controller is fastest by one to three orders over the conventional program; the tensor controller sits between them and is never faster than in matrix form.

Across the spatial sweep the slowdown rises from 2.2× at a maximum active set of 13 to 53.7× at a maximum active set of 54. In the horizon sweep it is instead smallest at the longest horizon, 4.6× at $N = 81$, the configuration where E3 found compression largest. There the matrix-form $\boldsymbol{S}$ has grown to about two gigabytes, so the matrix-form controller becomes memory-bound and slows to 6.8 milliseconds per step, while the compact tensor controller does not (figure 5.19).

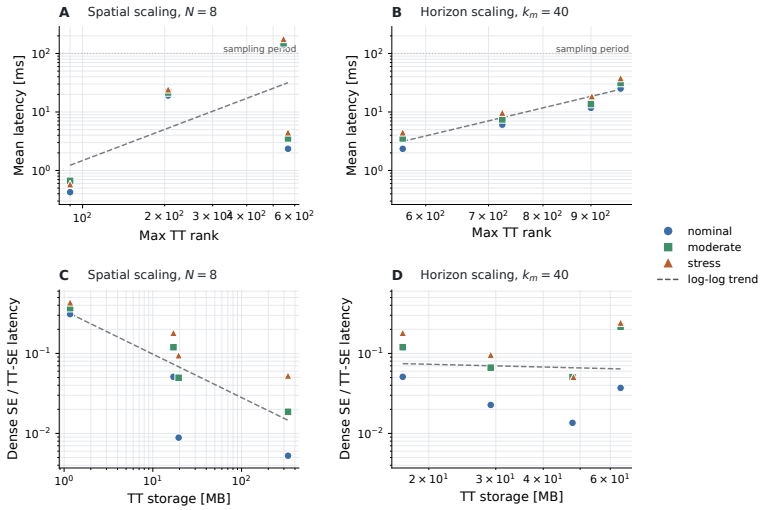


Figure 5.19: E5: the tensor controller's latency penalty against operator scale. The penalty is largest at moderate sizes and smallest in the long-horizon, storage-bound regime, where the matrix-form controller loses its cache advantage.

Analysis. Three quantities appear in this analysis and play distinct roles. The total parameter count $\sum_k R_{k-1} I_k J_k R_k$ sets the cost of one contraction (section 4.8). The maximum rank influences that count but is not identical to it, since a shape plan can raise the rank while lowering the count. The active-set size sets how many contractions each step performs.

At each online step the tensor controller applies its operators by contracting the cores, paying the parameter count per contraction. The operations on $\boldsymbol{S}$ and $\boldsymbol{M}$ repeat this work for each active constraint, so the per-step cost grows with the active-set size, while the matrix-form controller reads the entries it needs from stored arrays. Because the parameter count is the same quantity that sets the storage in section 5.7, compression does not trade storage for latency. A shape plan that shrinks the storage shrinks the online cost with it. E3's shape-plan variation confirms this. Splitting the physical modes raises the TT rank sevenfold to ninefold but leaves the latency unchanged, because the mode products shrink as the rank grows.

The slowdown relative to the matrix form is driven by the active-set size, not by the rank, as section 4.10 predicted. The two can be separated by fitting each against the measured slowdown. Across the horizon sweep the TT rank of $\boldsymbol{S}$ grows from 567 to 969, yet the slowdown shows no dependence on it, with a log-log fit giving $\propto R_{\max}(\boldsymbol{S})^{-0.14}$ at $R^2 = 0.003$. The active-set size predicts it almost exactly. Across the spatial sweep the slowdown scales as $|\mathcal{A}|_{\max}^{1.9}$ at $R^2 = 0.93$, close to quadratic in the maximum active set. The exponent is consistent with the structure of the loop. Reaching an active set of size $|\mathcal{A}|$ takes about $|\mathcal{A}|$ iterations, because each iteration adds or drops one constraint (Algorithm 3), and each iteration repeats active-set-sized work on the cores, extracting the active block $\widetilde{\mathcal{S}}_{\mathcal{A}\mathcal{A}}$ by index selection and contracting the active columns. Every such operation pays the core-chain work and the interpreter overhead that the matrix-form controller avoids by reading stored entries, so the repeated work compounds close to quadratically.

The three quantities thus separate cleanly. The total parameter count sets the tensor controller's absolute per-step expense, and it equals the storage, so a more compact operator is cheaper both to store and to apply. The rank affects latency only through that count, which is why normalising the latency by the rank collapses the size trend while the slowdown stays flat in the rank (figure 5.20). The active-set size sets the gap to the matrix-form controller, which pays for the same operations by reading stored entries instead of repeating core-chain work. The slowdown therefore reflects the active-set work and the loss of the matrix-form cache advantage, not the cost of compression, and the tensor controller is most useful at long horizons with compact operators, where the storage gain is largest and the slowdown smallest.

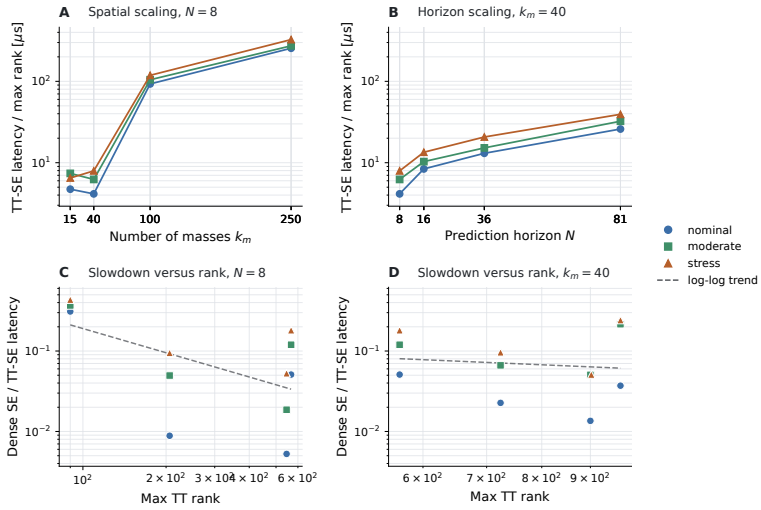


Figure 5.20: E5: rank-normalised latency (top) and speedup over the matrix form against maximum TT rank (bottom). Normalising by rank collapses the size trend, and the speedup is flat in rank, so the online penalty tracks the active-set work and the access cost, not the rank.

Scope. The absolute size of the slowdown depends on the implementation. The matrix-form controller runs its matrix-vector products on optimised linear-algebra kernels, while the tensor controller evaluates its contractions in Python loops and pays the interpreter's overhead at every step. Running those contractions as compiled code, on the same kind of kernels the matrix-form controller uses, would shrink the gap. The ordering and the scaling would remain. The matrix-form controller would still sit below the tensor controller, and the slowdown would still grow with the active-set size, because both follow from the model and not the implementation. The scaling fit has its own scope. The comparison keeps only the runs that pass the active-set work guard of section 5.3.2, and the dropped runs concentrate in the stress workloads with the largest active sets. The near-quadratic law is therefore established on runs where the two controllers do the same active-set work.

5.10. Relation to established MPC methods

The experiments above benchmark the tensor controller against the matrix-form SE controller and a single online QP solver. This section places it among the established approaches to the computational cost of constrained MPC. These approaches differ in where they pay these costs, offline, in storage, or online, and in how they scale with the problem (table 5.3). Explicit MPC is included as the extreme that precomputes the entire control law, whose offline and storage cost grow combinatorially and therefore do not scale. The placement is qualitative, drawn from the costs measured in this thesis and the asymptotic costs reported in the literature, and is used to point to directions for future work.

Table 5.3: Tensorised SE MPC among established approaches to the computational cost of MPC.

	Cost			
Method	Offline build	Storage	Online per step	Preferable when
Online QP [18], [45]	none	problem data only	iterative QP solve; grows with size and conditioning	no offline budget; time-varying model
Explicit MPC [6], [7]	$\mathcal{O}(2^{Np})$ (mpQP)	$\mathcal{O}(2^{Np})$ (region partition)	region lookup, logarithmic-time [21]	low state dimension; short horizon
Sparse / Riccati interior-point [11], [20], [46]	none	$\mathcal{O}(N)$	$\mathcal{O}(N)$ per iteration [19]	long horizon; online solve acceptable
Matrix-form SE [10]	forms $\boldsymbol{S} \in \mathbb{R}^{Np \times Np}$	$\mathcal{O}(Np^2)$	active-set; reads $\boldsymbol{S}$	while $\boldsymbol{S}$ fits in memory
Tensorised SE	$\boldsymbol{M}$-solve dominates	$\propto Np^{0.57}$	active-set on compressed cores	storage-bound, long horizon

The tensorised SE method occupies a narrow position. Like online QP and the banded solvers, it needs no precomputed state-space partition, so its storage grows far more slowly than that of explicit MPC [6]. Like the matrix-form SE controller, it keeps the precomputed operator set and needs no online matrix factorisation, so it inherits the one-to-three order online advantage over a conventional QP solver (section 5.9). What sets it apart is that it compresses the dual-coupling matrix $\boldsymbol{S}$ rather than avoiding it. Its advantage is storage at long horizons, and its weakness is the multiple-RHS construction cost of section 5.8.

The closest competitors are the sparse, partially condensed, and Riccati-based interior-point methods [11], [20], [46]. These exploit the stage-wise banded structure of the problem to reach a per-iteration cost linear in the horizon, without ever forming the matrix-form $\boldsymbol{S}$ [19]. They therefore avoid the very operator that the tensor pipeline spends its effort compressing, and that is the source of the construction bottleneck in section 5.8. This contrast is the clearest pointer to future work. Carrying the banded, stage-wise structure into the inverse-action solve could keep that solve low-rank throughout, not only in its result, and so attack the bottleneck at its source. The horizon-driven conditioning of $\boldsymbol{S}$ seen in section 5.6 invites a complementary horizon-independent preconditioner [37] for that solve.

Turning this qualitative placement into evidence would require a head-to-head study on the same problems and hardware, comparing the tensor controller against the banded solvers that skip $\boldsymbol{S}$ [19] and against embedded, code-generated active-set implementations [4], [45].

5.11. Synthesis

Table 5.4 maps each experiment to the question it supports. E0 fixes the reference. E1 and E2 answer SQ1. E3, E4, and E5 answer SQ2, SQ3, and SQ4.

Table 5.4: Evidence map: each experiment, the supporting question it addresses, and the claim it establishes.

Stage	Question	Claim established
E0	reference (underpins SQ1–SQ4)	matrix-form SE MPC reproduces the online QP to solver precision
E1	SQ1 (accuracy)	the target δ is a predictable, route-independent control parameter
E2	SQ1 (admissibility)	the admissible target is set by active-set boundary sensitivity, with closed-loop constraint satisfaction as the limiting criterion
E3	SQ2 (storage)	up to 36× compression at admissible accuracy, following the total parameter count, with the shape plan as the one free design choice
E4	SQ3 (construction)	no construction benefit; the multiple-RHS $\boldsymbol{M}$-solve is the bottleneck
E5	SQ4 (latency)	slower than in matrix form, but the penalty is smallest where the storage gain is largest

Accuracy comes first, because it fixes the operating point for every later measurement. E1 shows that the requested target δ is a predictable, route-independent control parameter (section 5.5). E2 then shows that the binding criterion is closed-loop constraint satisfaction. The operator error of order δ propagates to the online solve, and near a constraint boundary it can flip an active-set decision, so the constraint-violation metric $v_{\max}$ is the first to cross its limit (section 5.6). The uniform operator error of E1 does not predict when this happens, so the admissible accuracy must be read from closed-loop constraint satisfaction rather than from the operator-error norm. The admissible target therefore depends on the workload and the horizon rather than on the system size, and it tightens as the horizon grows. Every cost figure below is reported at the single conservative target $\delta^\star$ of equation (5.1), the tightest any tested case demands, so the storage ratios are lower bounds. A case with a looser admissible target would compress further.

Final-operator compression reaches 36× at $\delta^\star$, concentrated in the dual-coupling matrix $\boldsymbol{S}$, with the storage cost following the total parameter count in equation (2.4). The gain is large and clean along the horizon but weak and non-monotone along the state dimension, so the method pays off when the horizon, not the state dimension, is the dominant scaling axis. The shape-plan variation shows that the physical-mode factorisation is the one design choice that sets the compression, with the rank fixed by the tolerance and the dimensions fixed by the system.

Construction in TT form does not inherit this compressibility. The multiple-RHS $\boldsymbol{M}$-solve passes through intermediate ranks that exceed the rounded result and set the peak cost. It also exhausts memory at the same cases where the final operator is most compressible, so at the scales where compression matters most the operator can only be built by the matrix-form route.

Online evaluation is slower than in matrix form, by a factor that tracks the active-set size rather than the TT rank. Both SE controllers still beat the online QP by one to three orders, so the SE reduction, not the representation, is what wins online. The latency penalty is smallest in the long-horizon, storage-bound regime, where the matrix-form $\boldsymbol{S}$ grows large enough to become memory-bound while the compact tensor controller does not. Storage and online access cost are the same total parameter count, so a configuration that stores in less memory also evaluates faster on the tensor evaluator.

The dual-coupling matrix $\boldsymbol{S}$, with its inverse-action factor $\boldsymbol{M}$, sets all three costs. It carries the approximation error (E1), holds most of the storage (E3), is accessed by the online evaluator (E5), and is the target of the construction solve (E4). Chapter 6 integrates these findings, separating the costs the shape plan controls from the construction cost it does not, and discusses when the method is worthwhile and how the $\boldsymbol{M}$-solve might be kept low-rank throughout.

6

Conclusions, Limitations, and Future Work

MPC delivers high-quality control under input and state constraints, but its cost at scale limits where it can be deployed. This thesis asked to what extent, and under what conditions, TT representation of the condensed SE MPC operators changes the storage, offline-construction, and online-latency cost of the controller. The answer is that it improves one of the three. The deployed operators compress at admissible accuracy. The negative result is that building them in tensor form is not cheaper than building them as matrices, and online evaluation does not overtake the matrix form within the tested range.

The dual-coupling matrix $\boldsymbol{S}$ and the inverse-action factor $\boldsymbol{M}$ drive all three costs, but through two different quantities. Storage and online cost both scale with the total parameter count of the representation, over which the shape plan is the one free design choice. Construction does not. It is set by the intermediate ranks the multiple-RHS inverse-action solve passes through. This chapter expands on these mechanisms, draws out their lessons, and sets out the directions they open for future work.

6.1. Accuracy and closed-loop feasibility (SQ1)

The SE operators can be compressed without losing closed-loop feasibility. The accuracy this requires is set by closed-loop constraint satisfaction. Constraint satisfaction is driven by the controller's active-set decisions, which the operator error influences only indirectly. Section 5.5 shows that a requested target δ produces a final-operator perturbation that is predictable and independent of the construction route, so δ is a usable control parameter. Section 5.6 then shows which value of δ keeps the controller admissible.

The admissible value is governed by the sensitivity of the active set at the constraint boundary. The operator error of order δ enters the online solve at each use of the compressed operators. Near a constraint boundary it can change one active-set decision, and the controller then commits to an input that violates a constraint. The maximum closed-loop constraint violation $v_{\max}$ is therefore the first metric to cross its limit in every accepted-to-rejected transition, ahead of the trajectory and cost metrics. The Frobenius operator error of section 5.5 does not predict this, because it is a uniform norm that does not record which entries are perturbed or how close the trajectory runs to a boundary.

The accuracy this demands depends on the workload and the horizon, not on the system size. It ranges from about 10^{-4} in the nominal case to about 10^{-10} in the long-horizon stress case. When few constraints are active, the perturbed solution stays interior and the constraint slack absorbs the operator error, which is why the nominal target is so loose. The target tightens as the horizon grows, because $\boldsymbol{S}$ and $\boldsymbol{M}$ both have a dimension equal to the stacked constraint count Np, so a longer horizon couples more constraints and a single perturbation disturbs more active-set decisions at once. This is consistent with the prediction of section 4.10 that condensing worsens the conditioning of $\boldsymbol{S}$ as the horizon grows. Spatial growth at a fixed horizon does not have this effect. It makes each per-step block larger but does not couple the constraints across more time steps, so a single perturbation still disturbs about the same number of active-set decisions.

The admissible target therefore shows no consistent trend with system size. The rest of the chapter uses the single conservative target $\delta^\star$, the tightest any tested case requires, which makes the storage and latency figures that follow lower bounds.

6.2. Storage (SQ2)

Storage is the cost that tensorisation improves. At the conservative target $\delta^\star$, every configuration compresses, with storage ratios from 1.4× to 36.3× (section 5.7). The gain is concentrated in the dual-coupling matrix $\boldsymbol{S}$, which holds between 68% and 85% of the stored parameters, so the compression of $\boldsymbol{S}$ is, to a close approximation, the compression of the whole controller.

The natural way to report this compression, the maximum rank, does not work here. In many TT works the maximum rank stands in for compression, which is fair when the external dimensions are fixed. Two features of this setting break that. First, the maximum rank only compares operators of the same external dimensions. The method is meant to run across system sizes and horizons, so those dimensions differ from one case to the next. A given maximum rank then means something different in each case, so it cannot be compared across cases. Second, the ranks are highly uneven across the cores of a single operator. The Kronecker-sum structure in the condensing keeps the ranks it produces bounded. The inverse action that forms $\boldsymbol{M}$ is far less structured and drives its ranks up. One operator therefore spans low-rank and high-rank cores, and the maximum rank reports only the largest of them.

The total parameter count avoids both problems, because it sums the contribution of every core and so reflects each core's own rank together with its mode sizes. The experiments confirm that the two quantities can move in opposite directions. Splitting the physical modes of $\boldsymbol{S}$ raises its maximum rank but cuts the parameter count to as little as 13% of the single-core baseline, and two spatial configurations with almost equal rank burden compress by 7.6× and 2.8× (section 5.7).

The parameter count is set by three quantities, and only one of them is free. The rank is set by the approximation target, which is held as loose as closed-loop feasibility allows, so it cannot be lowered further to save storage. The external dimensions are set by the system size and the horizon, which the problem fixes, with only the horizon offering some freedom. The shape plan, the way those dimensions are assigned to cores, is the one choice left to the designer. In a setting like this, where the operators differ in dimension and rank from one case to the next, no single fixed factorisation serves every case. The shape plan has to be chosen per case, and it strongly influences the compression achieved.

The freedom the shape plan offers is limited by arithmetic. The horizon can be chosen to factor into convenient sizes, so the temporal modes stay small as the horizon grows. The physical sizes are fixed by the system and admit only the splits their factors allow. This is why the two scaling axes compress differently. Along the horizon the TT rank of $\boldsymbol{S}$ grows only logarithmically while its dimension grows linearly (section 4.10), and the parameter count grows as $Np^{0.57}$ against the matrix-form Np^2, so the compression rises cleanly to 36× at the longest horizon. Along the spatial axis the compression is irregular, because it depends on how the fixed physical sizes happen to factor rather than on a free choice.

6.3. Construction (SQ3)

The compressibility of the deployed operators does not transfer to the cost of building them. No tensor-native route builds them more cheaply than the matrix form (section 5.8). Where the routes that compress during assembly complete, they run one to four orders of magnitude slower than the matrix form, and on the largest and longest-horizon cases they exhaust memory, at the same configurations where section 5.7 found the operators most compressible.

The cost sits in one step, the multiple-RHS inverse-action solve for $\boldsymbol{M}$. It accounts for between 96.1% and 99.9% of the construction time and holds the entire peak memory. The solve reaches a low-rank result, but it passes through intermediate iterates whose ranks run well above that result. The accuracy the inverse action requires must be held throughout the solve, not only at its end, so these intermediate ranks set the peak time and memory. A compressible final operator therefore does not imply a cheap construction, because the rounding tolerance bounds the final operators but not the ranks reached on the way to them.

The two horizon trends are connected. Section 6.2 showed that longer horizons make the final operators more compressible, through the geometric decay of their off-diagonal blocks. Section 6.1 showed that the same growth worsens the conditioning of $\boldsymbol{S}$. The decay sets the rank of the result. The conditioning sets the difficulty of the solve that produces it. A worse-conditioned system needs more solver work, and the iterates that work passes through carry higher ranks. The same horizon growth that makes the operators most compressible therefore makes their inverse action hardest. The failures of the tensor-native routes land at exactly the configurations where the format pays off most.

This is why the construction cost stands apart from the other two. Storage (section 6.2) and online cost (section 6.4) both follow the total parameter count, which the shape plan controls. Construction is dominated by the intermediate ranks of the inverse-action solve. The only route that reaches the compressed operators forms the matrix-form $\boldsymbol{S}$ and rounds it at the end, so the construction still passes through the full matrix. The format shrinks the operator that is stored and applied, but not the object that must be built to obtain it. Lowering the construction cost would require an inverse-action solve that stays low-rank throughout, not only in its result.

6.4. Online latency (SQ4)

Unlike construction, the online cost and the storage are governed by the same quantity. The tensor controller applies its operators with the same per-core contraction that sets their storage, so a shape plan that reduces the storage reduces the online cost with it.

This does not make the tensor controller faster than the matrix form. Across the tested range it is slower, by a median factor of 2 to 54, and never faster (section 5.9). The rank does enter the online cost, through the same parameter count that sets the storage, but that is the tensor controller's absolute expense, not its gap to the matrix form. The gap is set by the online active-set size and by the implementation. Each active constraint repeats the contractions on $\boldsymbol{S}$ and $\boldsymbol{M}$, while the matrix-form controller reads the entries it needs from stored arrays, so the slowdown grows close to quadratically with the active-set size and is flat in the rank within the tested range. Part of the absolute gap comes from the interpreted evaluator, which runs the contractions in Python loops, so compiling them would narrow the gap without changing how it scales.

The gap is smallest where the storage gain is largest. At the longest horizon the slowdown falls to 4.6×, the regime where section 5.7 found the compression greatest. There the matrix-form $\boldsymbol{S}$ grows to about two gigabytes and becomes memory-bound, while the compact tensor controller does not. This points to a use case for the method. Where $\boldsymbol{S}$ no longer fits in fast memory, the tensor evaluator would become the better choice. The tested range ends just before that crossover. Both controllers beat a conventional online QP by one to three orders of magnitude across the range, but the SE reduction, not the representation, is what makes either fast online.

6.5. Answer to the research question

Taken together, these results answer the research question. TT representation of the condensed SE operators improves one of the three costs and worsens the other two. It compresses the deployed operators at admissible accuracy. It does not make them cheaper to build, and it does not make the online evaluation faster than the matrix form within the tested range. Within that range it trades a storage saving for a higher construction and online cost.

The method becomes the better choice under one condition. When $\boldsymbol{S}$ no longer fits in fast memory, the matrix-form controller becomes memory-bound while the compact tensor controller does not, so the tensor controller would then be both smaller and faster. This regime lies beyond the range the matrix-form baseline and the tensor-native constructors reach here, but the long-horizon results point toward it, where the slowdown is smallest and the compression largest. The condition in the research question is therefore a storage-bound one, in which the deployed operator is too large to hold and apply in matrix form. Within that condition the results already support a concrete deployment route. The operators are built by the matrix-form pipeline, compressed by the late-stage route, and run by the tensor-compatible evaluator. For long-horizon or large-system controllers whose operators outgrow fast memory, this hybrid is the practical form of the method.

Two changes would widen this regime. Compiling the online evaluator would remove the interpreter overhead that inflates the online gap, without changing how it scales. If a structure can be found that keeps the inverse-action solve low-rank throughout, the tensor-native construction could inherit the compressibility it now loses, and reach the sizes where the storage gain matters most. Neither is demonstrated here. This thesis therefore establishes where TT representation of these operators is worth using, the storage-bound long-horizon regime, and identifies the multiple-RHS inverse-action solve as the obstacle to extending it further.

6.6. Limitations

Offline, compression is controlled by the operator accuracy δ. Online, the controller must remain closed-loop admissible. The methodological choice the study rests on is to use δ as the proxy for that admissibility. It is convenient, because it is set directly during construction and gives a straightforward accept-or-reject test in closed loop. Its weakness is that it is a uniform operator-error norm, while the failure it must predict is not uniform. Admissibility is lost through a single active-set decision flipping near a constraint boundary (section 5.6), so it depends on which entries of the operator are perturbed and how close the trajectory runs to that boundary. The operator accuracy captures neither. It averages the perturbation over all entries and carries no information about the trajectory. A more informative bridge would link the operator error directly to the active-set sensitivity that governs the failure, rather than to the average perturbation. The results are internally consistent under operator accuracy, but such a bridge could move the quantitative conclusions, most of all the admissible target $\delta^\star$, which depends most strongly on that sensitivity. The admissible target also sets the storage and the online latency, so a bridge that relaxed it would lower both.

The benchmark is a single structured family, the mass–spring–damper chain of section 5.2. It was chosen because it is LTI and scales along two independent axes. The compression and cost laws are established on this family and are not claimed for arbitrary systems. They rest on two structural properties. The temporal factorisation relies on the block-Toeplitz structure of the condensed operators, which holds for any time-invariant prediction model and is lost under time-varying dynamics. The compression relies on the geometric decay of the off-diagonal blocks, which holds only for a stable plant. An unstable plant keeps the structure but loses the low rank. TT ranks are also hard to predict from the problem alone, so the numerical results are specific to this benchmark. The thesis therefore emphasises the mechanisms behind the results rather than the numbers themselves.

The acceptance rule of section 5.6 is empirical. It was measured over eight closed-loop steps and five seeds per workload, and it is not a recursive-feasibility or stability guarantee. It establishes that the compressed controller behaves admissibly on the tested trajectories, but it does not prove admissible behaviour for every initial condition, and it gives no formal analysis of closed-loop stability or recursive feasibility under compression.

The study benchmarks against two references, the matrix-form SE controller and a single online QP solver. It does not compare against the wider range of scalable MPC methods, such as sparse or partially condensed solvers, Riccati-based and first-order methods, or embedded implementations. The claims are therefore positioned relative to the SE pipeline, not against the whole solver landscape.

The remaining limitations are choices rather than fixed constraints, and a later study could relax them. Every case was compressed to the one conservative target $\delta^\star$, so the storage ratios are lower bounds, and a case with a looser admissible accuracy would compress further. Only one shape plan per operator was swept, rather than the full space of layouts, which leaves the storage figures conservative and the best layout uncharacterised. The closed-loop window was kept short to keep the comparison controlled, which limits the resolution of the admissible target. The online evaluator uses Python loops for operations such as the contractions, which inflate the absolute latency gap without changing its dependence on the active-set size. Section 6.7 takes up each of these.

6.7. Recommendations for future work

Shape-plan optimisation. The shape plan is the only free design variable over storage, but its freedom is limited, since the splits it can reach are fixed by how the external dimensions factor. Optimising within that space is still the most direct way to improve the method. Because the shape plan sets both the storage and the online cost through the total parameter count, optimising that count would lower both at once. The recommended study is a systematic search over layouts, taking the temporal–physical split used here as the baseline and the total parameter count as the objective. Whether any layout improves on this split is open, and the comparison should report the parameter count rather than the maximum rank, which section 5.7 showed to be an unreliable cost measure when a plan trades rank against mode sizes.

A low-rank multiple-RHS solve. The construction cost is set by the intermediate ranks of the multiple-RHS inverse-action solve, so the central direction is a solve that keeps those ranks low throughout, not only in its result. The column chunking of section 4.7 does not achieve this. It bounds the memory the intermediate iterates use, but not their rank, so a method that lowers the rank itself is needed. Two structural leads are open. The condensed Hessian has the Kronecker-sum form derived in section 4.4. Whether that structure can be exploited inside the solve to keep the iterates low-rank is one direction worth investigating. A block formulation could instead solve the constraint columns jointly, carrying the column index as an extra tensor mode, as Coulaud, Giraud, and Iannacito do for parameter-dependent RHSs with a TT variant of GMRES [25]. Their experiments meet the same obstacle in another form. The ranks of their intermediate iterates grow into the hundreds within a few iterations, so the joint formulation shares structure across columns but does not by itself keep the solve low-rank. If either kept the inverse-action solve low-rank throughout, the storage benefit of section 5.7 would become a construction benefit, and the method would extend beyond the storage-bound regime.

Parallel construction. Parallel execution would cut the wall-clock cost without changing the ranks. The column chunking already exposes concurrency at the column level, and the subproblems inside a chunk admit further parallelism along the alternating sweep. Distributing the chunks across cores or accelerators would lower the construction time and raise the effective per-process memory, whose shortage is the proximate cause of the out-of-memory failures on the largest cases. It complements a low-rank solver rather than replacing it, because it does not change the per-column intermediate rank.

A closed-loop accuracy criterion. The accuracy target is set conservatively, and a direct closed-loop criterion would relax it. The admissible target is read from a uniform operator error and then fixed at the worst case, yet section 5.6 showed that admissibility is governed by the active-set sensitivity at the constraint boundary, not by the average operator accuracy. A criterion that bounds the closed-loop constraint violation directly would relax the accuracy away from the boundary and tighten it only where it matters, replacing the single conservative $\delta^\star$ with a per-configuration target. Because the target sets the storage and the online latency, a looser per-configuration target would lower both.

Two studies would support this. The first is a survey of how often a bounded operator perturbation flips an active-set decision, as a function of the perturbation size and the distance to the boundary, which would identify what drives the misclassification. The second is a formal counterpart to the empirical acceptance rule of section 5.6, an analysis of closed-loop admissibility under a bounded operator perturbation that would replace the tested-trajectory result with a provable one.

Near-term engineering. Two engineering changes would make the method useful in the short term, without solving the open problems above. The first is to compile the online evaluator. The contractions now run in interpreted Python loops, which inflate the absolute latency gap (section 5.9). Running them as compiled kernels, of the kind the matrix-form controller already uses, would narrow the gap without changing its dependence on the active-set size. In the storage-bound regime, where the matrix-form controller is memory-bound, this could be enough to make the tensor controller the faster option.

The second is to accept the construction bottleneck rather than remove it. The matrix-form route builds the operators efficiently, and the tensor format deploys them efficiently, so the hybrid pipeline of matrix-form construction followed by final-stage compression is already a viable deployment path. Optimising it, with hardware-aware matrix-form construction and compression of only the operators for which the tensor format helps most, is the shortest route to deployment in the near term.

Simpler compression formats. The storage-bound regime also invites simpler compression than the TT. The gain of section 6.2 is that $\boldsymbol{S}$ no longer has to be stored and read in full, and any representation with a smaller parameter count can claim that same gain. A truncated SVD of $\boldsymbol{S}$ is the simplest candidate [47]. It stores two thin factors, and the online evaluator can then apply them and extract entries with the same compiled kernels the matrix-form controller uses. That removes the contraction and extraction overhead behind the slowdown of section 5.9. Whether the SVD compresses enough is an open empirical question. The tensor format exploits the decay of the off-diagonal blocks under a multilevel split, and that decay does not imply a small global rank. Hierarchical low-rank formats target the off-diagonal decay directly and sit between the two in complexity. The stored operators of section 5.7 already provide the data for this comparison, so the study is cheap. If a simpler format matches the compression at the admissible accuracy, the deployment case for the TT weakens. If it falls short, the case for the TT in the storage-bound regime is strengthened.

A study of TTM arithmetic. A study of TTM arithmetic on its own would test how far the shape-plan findings generalise. Characterising the cost of constructing, contracting, and rounding a TTM as a function of its mode shape, ranks, and external dimensions would show whether the shape-plan effect holds beyond this operator family. It would also indicate whether the non-monotone spatial compression of section 5.7 comes from the structure of $\boldsymbol{S}$ or from the underlying dynamics.

Such a study would also reach beyond this application. TT arithmetic can make computation much more efficient in the right setting, and the condensed MPC operators studied here are not that setting. Examining where it pays off in general, rather than in a single application, could yield results that reach beyond MPC.

References

[1] J. H. Lee, "Model predictive control: Review of the three decades of development," *International Journal of Control, Automation and Systems*, vol. 9, no. 3, pp. 415–424, 2011. DOI: 10.1007/s12555-011-0300-6.

[2] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5–6, pp. 1327–1349, 2021. DOI: 10.1007/s00170-021-07682-3.

[3] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014. DOI: 10.1016/j.automatica.2014.10.128.

[4] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008. DOI: 10.1002/rnc.1251.

[5] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides, "A condensed and sparse QP formulation for predictive control," in *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, 2011, pp. 5217–5222. DOI: 10.1109/CDC.2011.6160293.

[6] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002. DOI: 10.1016/S0005-1098(01)00174-1.

[7] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control: Towards New Challenging Applications*, ser. Lecture Notes in Control and Information Sciences, vol. 384, Berlin, Heidelberg: Springer, 2009, pp. 345–369. DOI: 10.1007/978-3-642-01094-1_29.

[8] A. Grancharova and T. A. Johansen, "Survey of explicit approaches to constrained optimal control," in *Switching and Learning in Feedback Systems*, ser. Lecture Notes in Computer Science, R. Murray-Smith and R. Shorten, Eds., vol. 3355, Berlin, Heidelberg: Springer, 2005, pp. 47–97. DOI: 10.1007/978-3-540-30560-6_3.

[9] I. Pappas et al., "Multiparametric programming in process systems engineering: Recent developments and path forward," *Frontiers in Chemical Engineering*, vol. 2, p. 620 168, 2021. DOI: 10.3389/fceng.2020.620168.

[10] F. Borrelli, M. Baoti, J. Pekar, and G. Stewart, "On the computation of linear model predictive control laws," *Automatica*, vol. 46, no. 6, pp. 1035–1041, 2010. DOI: 10.1016/j.automatica.2010.02.031.

[11] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *Proceedings of the European Control Conference (ECC)*, 2014, pp. 128–133. DOI: 10.1109/ECC.2014.6862490.

[12] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011. DOI: 10.1137/090752286.

[13] M. Oster, L. Sallandt, and R. Schneider, "Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats," *SIAM Journal on Scientific Computing*, vol. 44, no. 3, B746–B770, 2022. DOI: 10.1137/21M1412190.

[14] S. Dolgov, D. Kalise, and K. K. Kunisch, "Tensor decomposition methods for high-dimensional Hamilton–Jacobi–Bellman equations," *SIAM Journal on Scientific Computing*, vol. 43, no. 3, A1625–A1650, 2021. DOI: 10.1137/19M1305136.

[15] A. Gorodetsky, S. Karaman, and Y. Marzouk, "High-dimensional stochastic optimal control using continuous tensor decompositions," *The International Journal of Robotics Research*, vol. 37, no. 2–3, pp. 340–377, 2018. DOI: 10.1177/0278364917753994.

[16] P. GelSS, S. Klus, J. Eisert, and C. Schütte, "Multidimensional approximation of nonlinear dynamical systems," *Journal of Computational and Nonlinear Dynamics*, vol. 14, no. 6, p. 061 006, 2019. DOI: 10.1115/1.4043148.

[17] K. Batselier, Z. Chen, and N. Wong, "A tensor network Kalman filter with an application in recursive MIMO Volterra system identification," *Automatica*, vol. 84, pp. 17–25, 2017. DOI: 10.1016/j.automatica.2017.06.019.

[18] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. DOI: 10.1007/s12532-020-00179-2.

[19] G. Frison and M. Diehl, "HPIPM: A high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020. DOI: 10.1016/j.ifacol.2020.12.073.

[20] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, "Recent advances in quadratic programming algorithms for nonlinear model predictive control," *Vietnam Journal of Mathematics*, vol. 46, no. 4, pp. 863–882, 2018. DOI: 10.1007/s10013-018-0311-1.

[21] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003. DOI: 10.1016/S0005-1098(02)00308-4.

[22] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020. DOI: 10.1109/TCYB.2020.2999556.

[23] C. C. Yáñez, G. Pangalos, J.-H. Meyer, G. Lichtenberg, and J. S. Sáez, "Linear state signal shaping explicit model predictive control using tensor decompositions," *IEEE Access*, vol. 12, pp. 64 427–64 438, 2024. DOI: 10.1109/ACCESS.2024.3396352.

[24] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013. DOI: 10.1002/gamm.201310004.

[25] O. Coulaud, L. Giraud, and M. Iannacito, "A note on TT-GMRES for the solution of parametric linear systems," *Electronic Transactions on Numerical Analysis*, vol. 62, pp. 163–187, 2025. DOI: 10.1553/etna_vol62s163.

[26] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. DOI: 10.1137/07070111X.

[27] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends in Machine Learning*, vol. 9, no. 4–5, pp. 249–429, 2016. DOI: 10.1561/2200000059.

[28] V. de Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1084–1127, 2008. DOI: 10.1137/06066518X.

[29] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, A683–A713, 2012. DOI: 10.1137/100818893.

[30] S. V. Dolgov and D. V. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, A2248–A2271, 2014. DOI: 10.1137/140953289.

[31] A. Cichocki et al., "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *Foundations and Trends in Machine Learning*, vol. 9, no. 6, pp. 431–673, 2017. DOI: 10.1561/2200000067.

[32] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd. Madison, WI, USA: Nob Hill Publishing, 2017, ISBN: 978-0-9759377-3-0.

[33] J. M. Maciejowski, *Predictive Control with Constraints*. Harlow, UK: Prentice Hall, 2002, ISBN: 978-0-201-39823-6.

[34] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd. New York, NY, USA: Springer, 2006, ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[35] R. Fletcher, *Practical Methods of Optimization*, 2nd. Chichester, UK: John Wiley & Sons, 1987, ISBN: 978-0-471-91547-8. DOI: 10.1002/9781118723203.

[36] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, no. 1, pp. 1–33, 1983. DOI: 10.1007/BF02591962.

[37] I. McInerney, E. C. Kerrigan, and G. A. Constantinides, "Horizon-independent preconditioner design for linear predictive control," *IEEE Transactions on Automatic Control*, vol. 68, no. 1, pp. 580–587, 2023. DOI: 10.1109/TAC.2022.3145657.

[38] V. A. Kazeev, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, A1511–A1536, 2013. DOI: 10.1137/110844830.

[39] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd. Philadelphia, PA, USA: SIAM, 2003, ISBN: 978-0-89871-534-7. DOI: 10.1137/1.9780898718003.

[40] M. H. Gutknecht, "Block Krylov space methods for linear systems with multiple right-hand sides: An introduction," in *Modern Mathematical Models, Methods and Algorithms for Real World Systems*, A. H. Siddiqi, I. S. Duff, and O. Christensen, Eds., New Delhi, India: Anamaya Publishers, 2007, pp. 420–447.

[41] S. Demko, W. F. Moss, and P. W. Smith, "Decay rates for inverses of band matrices," *Mathematics of Computation*, vol. 43, no. 168, pp. 491–499, 1984. DOI: 10.1090/S0025-5718-1984-0758197-9.

[42] M. Benzi and V. Simoncini, "Decay bounds for functions of Hermitian matrices with banded or Kronecker structure," *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 3, pp. 1263–1282, 2015. DOI: 10.1137/151006159.

[43] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd. Philadelphia, PA, USA: SIAM, 2002, ISBN: 978-0-89871-521-7. DOI: 10.1137/1.9780898718027.

[44] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010. DOI: 10.1109/TCST.2009.2017934.

[45] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014. DOI: 10.1007/s12532-014-0071-1.

[46] D. Axehill, "Controlling the level of sparsity in MPC," *Systems & Control Letters*, vol. 76, pp. 1–7, 2015. DOI: 10.1016/j.sysconle.2014.12.002.

[47] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011. DOI: 10.1137/090771806.

A

Tensor Solver and Algorithm Details

This appendix gives the reproducible detail for the two pieces of tensor-native machinery that chapter 4 describes at the level of their contribution: the multiple-RHS inner solve of section 4.7 and the selected-column contraction of the online evaluator (section 4.8). The conceptual role of each is given there. What follows is the algorithmic detail and the per-phase cost.

A.1. Multiple-RHS column-chunked block-PCG

The outer solver of Algorithm 6 reduces the inverse-action systems $\widetilde{\mathcal{H}}\widetilde{\mathcal{L}} = \widetilde{\mathcal{F}}$ and $\widetilde{\mathcal{H}}\widetilde{\mathcal{M}} = \widetilde{\mathcal{G}}^{\top}$ to a sequence of local SPD systems, one per core. For the single-RHS factor $\widetilde{\mathcal{L}}$ each local system is an ordinary SPD solve. For the multiple-RHS factor $\widetilde{\mathcal{M}}$, whose Np columns share the same operator $\widetilde{\mathcal{H}}$ but carry one RHS per constraint, the local system is a multi-column solve, and forming all columns at once would set the peak memory by the largest intermediate rank across every column. Algorithm 8 processes the columns in chunks, running matrix-free PCG on a block of columns at a time and rounding between iterations, so the peak memory is set by the chunk rather than by the full column count. The operator $\widetilde{\mathcal{H}}$ is never formed. Only its action on a TT block is needed.

Algorithm 8 Multiple-RHS column-chunked block-PCG for the local SPD system [29], [39].

Require: local SPD operator $\boldsymbol{H}_k$ (as an action), RHSs $\boldsymbol{B}$ (Np columns), tolerance ε, chunk size c
Ensure: approximate solution $\boldsymbol{X}$ with $\|\boldsymbol{H}_k\boldsymbol{X} - \boldsymbol{B}\|_F \le \varepsilon\|\boldsymbol{B}\|_F$
1: partition the columns of $\boldsymbol{B}$ into chunks of size c
2: **for** each chunk $\boldsymbol{B}_J$ **do**
3: run PCG in the TT format, using only the action of $\boldsymbol{H}_k$
4: round the intermediate iterates to bound the rank growth across the chunk
5: **end for**
6: assemble $\boldsymbol{X}$ from the chunk solutions

The chunking bounds the memory of each local solve, but it does not remove the work. Each chunk still drives its residual down by an iterative sweep whose intermediate ranks are set by the problem, so the cost that remains after the mitigation is the structural floor of the multiple-RHS inverse action, which section 5.8 identifies as the construction bottleneck.

A.2. Selected-column contraction in the online evaluator

The online evaluator (Algorithm 7) needs products of the operators with a vector supported only on the active set, $\widetilde{\mathcal{S}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$ and $\widetilde{\mathcal{M}}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$. Contracting each active column separately would recompute the partial contractions that columns sharing a mode-index prefix have in common. Algorithm 9 groups the active columns by shared prefix and computes each shared partial contraction once, so the cost scales with the number of distinct prefixes rather than with the active-set size times the chain length.

Algorithm 9 Prefix-grouped selected-column contraction.

Require: TTM operator $\boldsymbol{O}$, active index set $\mathcal{A}$, multipliers $\boldsymbol{\lambda}_{\mathcal{A}}$
Ensure: $\boldsymbol{O}_{:,\mathcal{A}}\boldsymbol{\lambda}_{\mathcal{A}}$
1: split each active column index into its per-mode multi-index (Definition 2.5)
2: group the active columns by shared mode-index prefix
3: **for** each core, from the shared end of the cores outward **do**
4: contract the partial product once per distinct prefix, reusing it for every column in the group
5: **end for**
6: combine the per-column contractions weighted by $\boldsymbol{\lambda}_{\mathcal{A}}$

A further reduction applies because the receding-horizon law uses only the first input. When only u_0 is required, only the first m rows of $\widetilde{\mathcal{L}}$ and $\widetilde{\mathcal{M}}$ are contracted, not the whole horizon.

A.3. Per-phase cost

Each operator access in the online evaluator is a sequence of core contractions, with cost equal to the total parameter count $\sum_k R_{k-1}I_kJ_kR_k$ over the cores, bounded by $\mathcal{O}\big(dR^2IJ\big)$ in the maximum rank R and largest mode size IJ. The active-set block operations on $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{M}}$ scale in addition with the active-set size, and the reduced dual system is the matrix-form $q \times q$ solve at the active-set size q. As section 4.8 notes, this access cost is the same sum $R_{k-1}I_kJ_kR_k$ over the cores that sets the storage of equation (2.4), so a factorisation that reduces storage reduces the access cost in step with it.

The construction cost is governed by a different quantity. The inverse-action solve passes through intermediate ranks larger than those of its rounded result, and the rounding that controls rank growth costs on the order of the cube of the TT rank at each operation, so the peak construction cost is set by the largest intermediate rank, not by the final rank. This is the intermediate-rank versus final-rank distinction that section 5.8 measures. The final operators are bounded by the rounding tolerance, but the intermediate ranks the multiple-RHS solve visits are not, and they set the bottleneck.

B

Experimental Configurations and Reproducibility

This appendix collects the configuration set, the tolerance and acceptance schedule, and the software environment behind chapter 5, so that the reported results can be reproduced from the data files listed in Appendix B.5. The accompanying code repository `https://github.com/smaljers01/TT-SEMPC` contains the scripts and data files referenced throughout.

B.1. Configuration set

The benchmark is the mass–spring–damper chain of section 5.2, scaled along the two independent axes of section 5.2.1. Seven distinct configurations span the two sweeps (table B.1). The spatial sweep grows the chain length k_m at fixed horizon $N = 8$. The horizon sweep grows N at fixed chain length $k_m = 40$. The two sweeps share the base configuration C2. Each chain length k_m fixes the state, input, and per-step constraint dimensions through $n = 2k_m$, m the actuated masses, and $p = 2(n+m)$, and the dual dimension is $Np = Np$.

Table B.1: The seven benchmark configurations. C2 is the base shared by both sweeps; the spatial sweep is C1, C2, C6, C7 at $N = 8$, and the horizon sweep is C2, C3, C4, C5 at $k_m = 40$.

Config	Sweep	k_m	n	m	p	N	Np
C1	spatial	15	30	8	76	8	608
C2	spatial/horizon	40	80	20	200	8	1600
C3	horizon	40	80	20	200	16	3200
C4	horizon	40	80	20	200	36	7200
C5	horizon	40	80	20	200	81	16200
C6	spatial	100	200	50	500	8	4000
C7	spatial	250	500	125	1250	8	10000

The result files of Appendix B.5 index each configuration by a `config_key` string rather than by its C-label. The two correspond as C1 = `small_N_8`, C2 = `medium_N_8`, C3 = `medium_N_16`, C4 = `medium_N_36`, C5 = `medium_N_81`, C6 = `large_N_8`, and C7 = `very_large_N_8`, so any row of the extended-results tables in Appendix C can be traced to the records that produced it.

B.2. Workloads

Constraint activity is set by how close the initial state sits to the constraint boundary, parameterised by the fraction α of equation (5.5). Three workload classes are used: nominal ($\alpha = 0.25$), moderate ($\alpha = 0.6$), and stress ($\alpha = 0.8$). Five seeds are drawn per class, and each case is run for eight receding-horizon closed-loop

steps, giving $7 \times 3 \times 5 = 105$ controller cases per experiment that uses the full grid. E2 additionally sweeps the nine accuracy targets per case, giving the 315 cases per workload reported in section 5.6. The online QP reference of E0 and E5 is OSQP [18] on the same condensed program.

B.3. Tolerance and acceptance schedule

Every internal tolerance derives from a single final-operator target δ through the staged schedule $\varepsilon_p \le \varepsilon_c \le \varepsilon_s \le \varepsilon_a \le \varepsilon_f = \delta$ of equation (4.15), each stage at least as tight as those downstream and floored at machine precision. The cost experiments E3, E4, and E5 are all run at the single conservative target

$$\delta^\star = \min_{\text{config, workload}} \delta_{\text{safe}} \approx 10^{-10}, \tag{B.1}$$

the tightest admissible accuracy any tested case demands, certified by E2 (section 5.6). The acceptance thresholds of E2 are constraint satisfaction, $v_{\max} \le 10^{-4}$, as the primary criterion, with the input and state deviations required below 5% and the cost change below 1% as secondary quality checks, motivated in section 5.6. The admissible target δ_{safe} is the loosest δ such that it and every tighter tested value pass.

B.4. Software, hardware, and provenance

All operators and controllers are computed in double precision. The tensor-native construction routes of E4 are run under a 12 gigabyte resident-memory cap, and a route that exhausts the cap before reaching $\delta^\star$ is recorded as a memory failure rather than a slow success. Latency in E5 is measured under the repeated-timing protocol of section 5.3.2, with warm-up evaluations priming the caches before each step is timed repeatedly and the mean, 95th percentile, and maximum reported.

Each result record carries its own provenance, so a row can be traced to the exact code and policy that produced it. The runs in this thesis were produced at git commit `e99857e`, under solver policy `multicore_spd_mals_..._v6` and tolerance policy `direct_lmst_v4_target_1e10`, with the configuration digest, random seeds, and timing repetitions stored alongside every record.

B.5. Data files

The cleaned result records are the per-experiment `.jsonl` files of table B.2, one batch per experiment E0–E5 together with the shape-plan variation files behind the result of section 5.7. The table lists every file with its record count.

Table B.2: The cleaned result files and their record counts. Every per-configuration table in Appendix C is generated mechanically from these files, so each reported number traces to the records behind it.

Data file (`.jsonl`)	Exp.	Contents	Records
`E0_dense_reference`	E0	Uncompressed reference vs. OSQP: consistency, KKT, agreement	45
`E1_target_tracking`	E1	Target tracking across δ and the three routes	27
`E2_robustness`	E2	Closed-loop runs across the δ grid (per seed, workload)	945
`E2_thresholds`	E2	Per-case admissible target δ_{safe} and binding metrics	105
`E3_storage`	E3	Final-operator storage, ranks, per-operator decomposition	21
`E4_construction`	E4	Construction time and peak memory per route	28
`E4_offline_build`	E4	Per-build offline records behind E4	230
`E5_online_latency`	E5	Per-step online latency by method, workload, seed	315
`supporting_E6_temporal_split`	E6	Temporal-only factorisation contribution	8
`supporting_E7_dynamics_split`	E7	Physical-only factorisation contribution	8
`supporting_E8_both_split`	E8	Combined factorisation contribution	14
`supporting_E6-E8_split_latency`	E6–E8	Latency of the factorisation variants	30

The per-configuration tables of Appendix C are generated directly from these files by a single reporting script, rather than transcribed, so the summary and full tables there are reproducible from the records and cannot drift from them.

C

Extended Results and Data Tables

This appendix gives the per-configuration numbers behind the figures of chapter 5. It is organised by experiment, E0 through E5 followed by the shape-plan variation, and every section pairs a summary table, the summary results the main text reads, with a full table that exposes the underlying per-case records so the result can be checked end to end. The seven configurations C1–C7 are those of table B.1. The mapping to the `config_key` strings used in the data files is given there. Every table body is generated mechanically from the result files of Appendix B.5.

C.1. Reading the tables

Three reporting conventions, stated once here, apply throughout. First, the cost experiments E3, E4, and E5 are all measured at the single conservative target $\delta^\star \approx 10^{-10}$ of equation (B.1), so a quantity that is a property of the compressed operators, such as a storage ratio or a construction time, does not depend on the seed or the workload and is reported as one value per configuration. The storage records confirm this invariance directly, the three workload replicates of each configuration agreeing exactly, up to a sub-percent truncation difference in the single case C4. Second, quantities that depend on the path through the active set, namely admissibility (E2) and online latency (E5), do vary with workload and seed and are reported either as the spread or as the median over the five seeds and three workloads of each configuration, as noted in each table. Third, where a worst case is reported, it is the maximum over all replicates of a configuration, so that a single bound covers every tested case at once. The admissible target δ_{safe} in particular is the loosest tolerance such that it and every tighter tested value also pass (section 5.6).

C.2. Reference validation (E0)

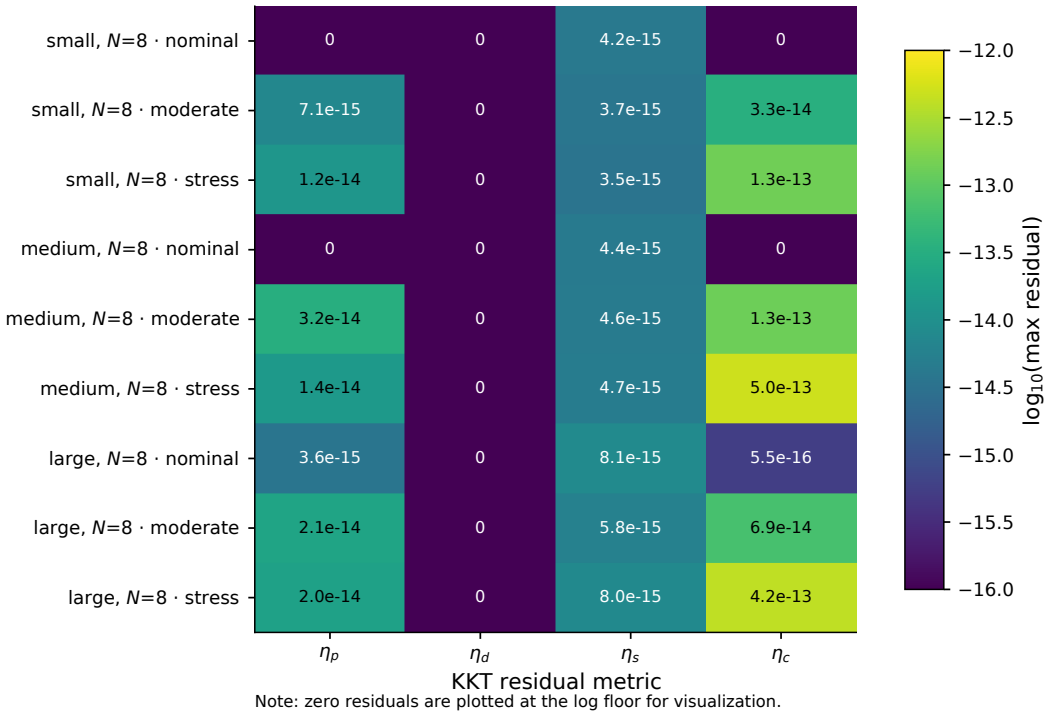
Table C.1 reports, for each configuration validated in E0 (section 5.4), the worst value over all workloads and seeds of the four operator-consistency residuals $\eta_{\boldsymbol{L}}, \eta_{\boldsymbol{M}}, \eta_{\boldsymbol{S}}, \eta_{\boldsymbol{T}}$ of equation (5.6), the objective and input-sequence errors e_J, e_U of equation (5.8), and the worst closed-loop violation $v_{\max}$. Every residual sits at the double-precision floor. The sole large figure is $e_U \approx 3.7 \times 10^{-7}$, the active-set ill-conditioning analysed in section 5.4, not a pipeline error. Table C.2 breaks the optimality residuals out per workload, giving the four KKT residuals $\eta_s, \eta_p, \eta_d, \eta_c$ of equation (5.9) alongside the same agreement metrics, and figure C.1 shows the full residual distribution.

Table C.1: E0 reference validation: worst observed agreement and consistency metrics per configuration, over all workloads and seeds. All but e_U sit at the double-precision floor.

Config	Np	η_L	η_M	η_S	η_T	e_J	e_U	$v_{\max}$
C1	608	9×10^{-16}	2.9×10^{-14}	0	0	1.1×10^{-15}	3.7×10^{-7}	1.1×10^{-14}
C2	1600	1.6×10^{-15}	3.6×10^{-14}	0	0	1.1×10^{-15}	4.8×10^{-8}	3.2×10^{-14}
C6	4000	1.7×10^{-15}	4.9×10^{-14}	0	0	9.3×10^{-16}	1.4×10^{-7}	2.1×10^{-14}

Table C.2: E0 full transparency: worst-case KKT residuals (stationarity η_s, primal η_p, dual η_d, complementarity η_c) and agreement metrics per configuration and workload, over the five seeds.

Config	Workload	η_s	η_p	η_d	η_c	e_J	e_U	$v_{\max}$
C1	nominal	4.2×10^{-15}	0	0	0	1.1×10^{-15}	2.7×10^{-11}	0
C1	moderate	3.7×10^{-15}	7.1×10^{-15}	0	3.3×10^{-14}	9.5×10^{-16}	3.7×10^{-7}	7.1×10^{-15}
C1	stress	3.5×10^{-15}	1.2×10^{-14}	0	1.3×10^{-13}	1.1×10^{-15}	5.4×10^{-9}	1.1×10^{-14}
C2	nominal	4.4×10^{-15}	0	0	0	1.1×10^{-15}	2.7×10^{-11}	0
C2	moderate	4.6×10^{-15}	3.2×10^{-14}	0	1.3×10^{-13}	8.7×10^{-16}	4.8×10^{-8}	3.2×10^{-14}
C2	stress	4.7×10^{-15}	1.4×10^{-14}	0	5×10^{-13}	8.8×10^{-16}	3.1×10^{-8}	1.4×10^{-14}
C6	nominal	8.1×10^{-15}	3.6×10^{-15}	0	5.5×10^{-16}	9×10^{-16}	1.4×10^{-7}	3.6×10^{-15}
C6	moderate	5.8×10^{-15}	2.1×10^{-14}	0	6.9×10^{-14}	7.6×10^{-16}	8.3×10^{-8}	2.1×10^{-14}
C6	stress	8×10^{-15}	2×10^{-14}	0	4.2×10^{-13}	9.3×10^{-16}	4.2×10^{-8}	2×10^{-14}



Note: zero residuals are plotted at the log floor for visualization.

Figure C.1: E0 supporting: the four KKT residuals across all cases. All sit at the double-precision floor, confirming the matrix-form reference is exact and that the large input-sequence error of section 5.4 is active-set ill-conditioning, not a pipeline defect.

C.3. Target tracking and proxy validity (E1)

Table C.3 reports the E1 sweep of section 5.5 on the medium configuration C2. At each requested target δ it gives the achieved worst-operator error $e^{\max}$ of equation (5.12) for the late, mid, and early routes, the tracking ratio $\eta = e^{\max}_{\text{late}}/\delta$, and the maximum TT rank. The three routes agree to within about 10% from 10^{-4} to 10^{-11}, which is what justifies the late route as the proxy used in E2 and E3. Below 10^{-11} the mid and early routes floor at their iterative solver residual while the late route continues. Table C.4 gives the per-operator errors $e_{\boldsymbol{L}}, e_{\boldsymbol{M}}, e_{\boldsymbol{S}}, e_{\boldsymbol{T}}$ of the late route together with the rank and build time, showing that $\boldsymbol{S}$ and $\boldsymbol{M}$ carry most of the error, as figure 5.6 reports.

Table C.3: E1 target tracking on C2: achieved worst-operator error $e^{\max}$ by route, the tracking ratio η, and the maximum TT rank, against the requested target δ.

δ	$e^{\max}_{\text{late}}$	$e^{\max}_{\text{mid}}$	$e^{\max}_{\text{early}}$	$\eta = e^{\max}_{\text{late}}/\delta$	$R_{\max}$
10^{-4}	8.7×10^{-5}	8.5×10^{-5}	8.5×10^{-5}	0.87	114
10^{-5}	7.6×10^{-6}	8.7×10^{-6}	8.7×10^{-6}	0.76	179
10^{-6}	8.1×10^{-7}	8.9×10^{-7}	8.9×10^{-7}	0.81	244
10^{-7}	8.2×10^{-8}	8.3×10^{-8}	8.3×10^{-8}	0.82	311
10^{-8}	8.2×10^{-9}	9×10^{-9}	9×10^{-9}	0.82	390
10^{-9}	8.3×10^{-10}	8.8×10^{-10}	8.8×10^{-10}	0.83	476
10^{-10}	7×10^{-11}	6.6×10^{-11}	6.6×10^{-11}	0.70	567
10^{-11}	7.1×10^{-12}	6.9×10^{-12}	6.9×10^{-12}	0.71	666
10^{-12}	6.3×10^{-13}	3.3×10^{-12}	1.4×10^{-12}	0.63	769

Table C.4: E1 full transparency: per-operator relative error of the late route on C2 against the requested target δ, with the maximum TT rank and the total construction time.

δ	$e_{\boldsymbol{L}}$	$e_{\boldsymbol{M}}$	$e_{\boldsymbol{S}}$	$e_{\boldsymbol{T}}$	$R_{\max}$	build (s)
10^{-4}	4×10^{-5}	8.1×10^{-5}	8.7×10^{-5}	5.4×10^{-5}	114	0.9
10^{-5}	5×10^{-6}	7.6×10^{-6}	7.6×10^{-6}	6.1×10^{-6}	179	0.5
10^{-6}	6×10^{-7}	7.8×10^{-7}	8.1×10^{-7}	5.6×10^{-7}	244	0.7
10^{-7}	2.9×10^{-8}	7.2×10^{-8}	8.2×10^{-8}	5.8×10^{-8}	311	0.7
10^{-8}	4.1×10^{-9}	8.1×10^{-9}	8.2×10^{-9}	5×10^{-9}	390	0.7
10^{-9}	5.3×10^{-10}	7.1×10^{-10}	8.3×10^{-10}	4.5×10^{-10}	476	2.0
10^{-10}	5.6×10^{-11}	3.5×10^{-11}	7×10^{-11}	5.4×10^{-11}	567	2.3
10^{-11}	4.5×10^{-12}	4×10^{-12}	7.1×10^{-12}	3.2×10^{-12}	666	0.9
10^{-12}	5.6×10^{-13}	1.3×10^{-13}	6.3×10^{-13}	5.8×10^{-13}	769	0.8

C.4. Closed-loop admissibility (E2)

Table C.5 reports the admissible target δ_{safe} of section 5.6 per configuration and workload, the tightest over the five seeds. The horizon sweep C2–C5 tightens steeply under the stress workload, to about 10^{-10} at the longest horizon, while the spatial sweep stays far more permissive. The most restrictive case across the whole grid, C4 under stress at $\delta_{\text{safe}} \approx 10^{-10}$, sets the conservative target $\delta^\star$. Table C.6 is the transparency record behind the acceptance rule. For every configuration and workload it gives δ_{safe}, the worst accepted value of each of the four acceptance metrics, constraint violation ${v_{\max}}^{\text{acc}}$, input and state deviation $e_u^{\text{acc}}, e_x^{\text{acc}}$, and cost change ΔJ^{acc}, and the number of accepted cases out of fifteen. The constraint-violation margin stays below its 10^{-4} gate on every accepted case while the secondary metrics stay well inside their 5% and 1% bounds, which is the mechanism of section 5.6 seen case by case. A representative closed-loop trajectory is shown in figure C.2.

Table C.5: E2 admissible target δ_{safe} per configuration and workload, the tightest (most conservative) value over the five seeds. Entries are the achieved final-operator error at the loosest passing requested target. The minimum over the whole grid sets $\delta^\star$.

Config	N	Np	δ_{safe} (nominal)	δ_{safe} (moderate)	δ_{safe} (stress)
C1	8	608	7.4×10^{-5}	7.8×10^{-6}	7.8×10^{-6}
C2	8	1600	8.7×10^{-5}	8.1×10^{-7}	8.3×10^{-10}
C3	16	3200	9.5×10^{-5}	9.6×10^{-6}	8.6×10^{-10}
C4	36	7200	10^{-4}	9.6×10^{-7}	9.7×10^{-11}
C5	81	16200	10^{-4}	1.1×10^{-6}	1.1×10^{-10}
C6	8	4000	8.4×10^{-5}	7.6×10^{-7}	7.6×10^{-7}
C7	8	10000	10^{-4}	7.6×10^{-7}	8×10^{-8}

Table C.6: E2 full transparency: admissible target and the worst accepted value of each acceptance metric per configuration and workload, with the accepted count out of the fifteen seed–workload cases. The constraint-violation metric $v_{\max}^{\text{acc}}$ stays below its 10^{-4} gate throughout.

Config	Workload	δ_{safe}	${v_{\max}}^{\text{acc}}$	e_u^{acc}	e_x^{acc}	ΔJ^{acc}	accepted
C1	nominal	7.4×10^{-5}	0	8.1×10^{-7}	4.4×10^{-7}	1.5×10^{-8}	45/45
C1	moderate	7.8×10^{-6}	1.1×10^{-5}	5.4×10^{-6}	2.3×10^{-6}	1.7×10^{-8}	43/45
C1	stress	7.8×10^{-6}	8.5×10^{-5}	2×10^{-5}	10^{-5}	1.5×10^{-7}	43/45
C2	nominal	8.7×10^{-5}	0	4.6×10^{-5}	4.3×10^{-5}	1.2×10^{-6}	45/45
C2	moderate	8.1×10^{-7}	6.9×10^{-5}	8×10^{-5}	4.8×10^{-5}	3.1×10^{-7}	40/45
C2	stress	8.3×10^{-10}	2.4×10^{-5}	3.4×10^{-6}	2.3×10^{-6}	6×10^{-8}	34/45
C3	nominal	9.5×10^{-5}	0	6.9×10^{-5}	3.4×10^{-5}	2.8×10^{-7}	45/45
C3	moderate	9.6×10^{-6}	7×10^{-5}	1.9×10^{-5}	1.1×10^{-5}	2.2×10^{-7}	40/45
C3	stress	8.6×10^{-10}	5×10^{-5}	5.5×10^{-6}	3.7×10^{-6}	3.1×10^{-8}	32/45
C4	nominal	10^{-4}	0	7.4×10^{-5}	5.2×10^{-5}	3.6×10^{-7}	45/45
C4	moderate	9.6×10^{-7}	9×10^{-5}	2.1×10^{-4}	1.2×10^{-4}	8.2×10^{-7}	40/45
C4	stress	9.7×10^{-11}	7.2×10^{-5}	4.7×10^{-5}	2×10^{-5}	2.5×10^{-7}	32/45
C5	nominal	10^{-4}	0	7.4×10^{-5}	5.2×10^{-5}	3.8×10^{-7}	45/45
C5	moderate	1.1×10^{-6}	7.8×10^{-5}	3.9×10^{-4}	1.8×10^{-4}	5.1×10^{-6}	40/45
C5	stress	1.1×10^{-10}	9.8×10^{-5}	6.2×10^{-5}	4.2×10^{-5}	7.9×10^{-7}	32/45
C6	nominal	8.4×10^{-5}	1.7×10^{-5}	1.4×10^{-4}	7×10^{-5}	5.4×10^{-7}	45/45
C6	moderate	7.6×10^{-7}	5.2×10^{-5}	2.2×10^{-5}	1.1×10^{-5}	4.3×10^{-8}	39/45
C6	stress	7.6×10^{-7}	3.4×10^{-5}	1.5×10^{-5}	8.6×10^{-6}	3.4×10^{-8}	35/45
C7	nominal	10^{-4}	0	2.4×10^{-4}	1.8×10^{-4}	2.1×10^{-7}	45/45
C7	moderate	7.6×10^{-7}	7.7×10^{-5}	3.2×10^{-5}	1.9×10^{-5}	3.6×10^{-8}	39/45
C7	stress	8×10^{-8}	9.9×10^{-5}	9×10^{-5}	4.1×10^{-5}	1.1×10^{-8}	35/45

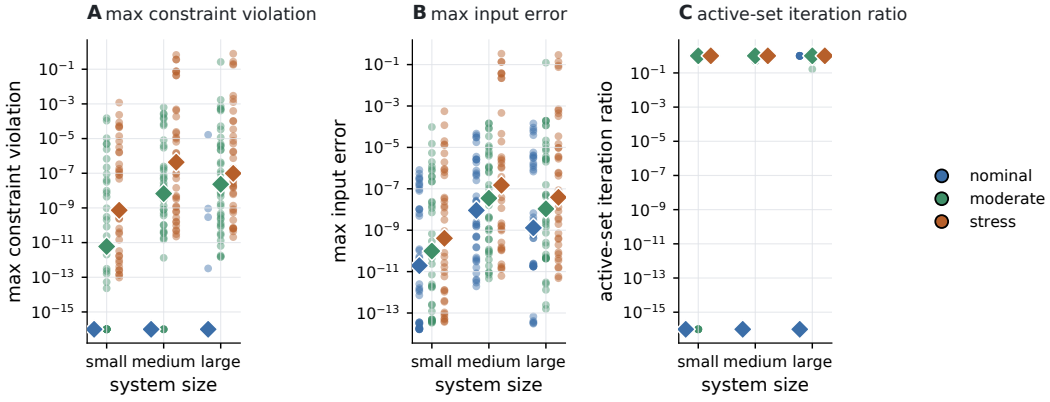


Figure C.2: E2 supporting: a representative closed-loop trajectory comparing the compressed controller against the matrix-form reference. Maximum constraint violation, maximum input error, and the active-set iteration ratio across all seeds and workloads stay within the admissible band throughout.

C.5. Storage compression (E3)

Table C.7 reports the storage result of section 5.7 per configuration at $\delta^\star$: the matrix-to-TT ratio ρ_{storage}, the maximum TT rank of the dual-coupling matrix $R_{\max}(\boldsymbol{S})$, the rank burden $R_{\max}(\boldsymbol{S})/Np$, and the worst operator error $e^{\max}$. The horizon sweep C2–C5 shows the clean monotone trend $1.4 \to 3.2 \to 9.7 \to 36.3\times$ that follows the falling rank burden. The spatial sweep is non-monotone, which section 5.7 traces to the tensorisation. Table C.8 resolves the budget per operator, giving each operator's relative error, matrix-form and TT storage, compression ratio ρ_O, share of the total storage ϕ_O, and maximum rank. The dual-coupling matrix $\boldsymbol{S}$ holds between 68% and 85% of the total storage throughout, which is why the total ratio tracks its compression. Figure C.3 shows the per-operator rank across both sweeps.

Table C.7: E3 storage per configuration at $\delta^\star$. The ratios are lower bounds, since every case is held to the single conservative target.

Config	N	Np	ρ_{storage}	$R_{\max}(\boldsymbol{S})$	$R_{\max}(\boldsymbol{S})/Np$	$e^{\max}$
C1	8	608	2.9×	57	0.094	6.2×10^{-11}
C2	8	1600	1.4×	567	0.354	7×10^{-11}
C3	16	3200	3.2×	724	0.226	8.8×10^{-11}
C4	36	7200	9.7×	900	0.125	9.7×10^{-11}
C5	81	16200	36.3×	969	0.060	1.1×10^{-10}
C6	8	4000	7.6×	206	0.051	8.4×10^{-11}
C7	8	10000	2.8×	546	0.055	7.9×10^{-11}

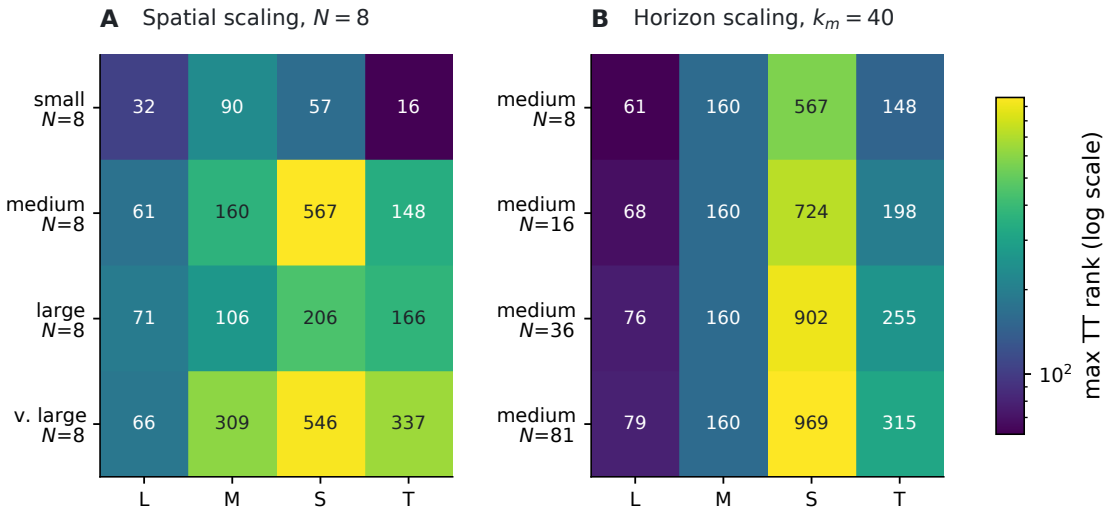


Figure C.3: E3 supporting: maximum TT rank per operator ($\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{T}$) across the spatial and horizon sweeps. The dual-coupling matrix $\boldsymbol{S}$ carries the highest TT ranks throughout, which is why it dominates both the total storage and the construction cost.

C.6. Construction cost (E4)

Table C.9 reports the total construction time of the matrix-form, late, mid, and early routes of section 5.8 at $\delta^\star$. The matrix-form and late routes complete throughout. The mid and early routes are one to four orders of magnitude slower where they complete, reaching about 2.3×10^4 times the matrix-form time on C3, and exhaust the 12 gigabyte cap before reaching $\delta^\star$ on the longest-horizon and largest-spatial cases C4, C5, and C7, marked †. Table C.10 localises the cost. For each route it gives the total time, the inverse-action solve time and its share of the total, the peak resident memory, and the route status. The inverse-action solve is between 96.1% and 99.9% of the construction time of every completed mid and early route and holds the entire peak memory, which is the bottleneck identified in section 5.8. The late route performs no inverse-action solve and so has no entry in that column. Figure C.4 plots time and memory relative to the matrix form.

Table C.8: E3 full transparency: per-operator relative error e_O, matrix-form and TT storage, compression ratio ρ_O, share of the total storage ϕ_O, and maximum rank, at $\delta^\star$. The offset $\bar{\boldsymbol{w}}$ is carried in vector form. Storage is in megabytes of double-precision parameters.

Config	Op.	e_O	uncompressed (MB)	TT (MB)	ρ_O	ϕ_O	$R_{\max}$
C1	$\boldsymbol{L}$	8.7×10^{-15}	0.02	0.02	0.63×	0.021	32
C1	$\boldsymbol{M}$	6.2×10^{-11}	0.31	0.27	1.16×	0.229	90
C1	$\boldsymbol{S}$	2.7×10^{-11}	2.96	0.79	3.72×	0.679	57
C1	$\boldsymbol{T}$	3.8×10^{-15}	0.15	0.08	1.87×	0.067	16
C1	$\bar{\boldsymbol{w}}$	–	0.00	0.00	1.00×	0.004	–
C2	$\boldsymbol{L}$	5.6×10^{-11}	0.10	0.12	0.86×	0.007	61
C2	$\boldsymbol{M}$	3.5×10^{-11}	2.05	1.93	1.06×	0.113	160
C2	$\boldsymbol{S}$	7×10^{-11}	20.48	13.87	1.48×	0.812	567
C2	$\boldsymbol{T}$	5.4×10^{-11}	1.02	1.14	0.90×	0.067	148
C2	$\bar{\boldsymbol{w}}$	–	0.01	0.01	1.00×	0.001	–
C3	$\boldsymbol{L}$	3.9×10^{-11}	0.20	0.19	1.09×	0.006	68
C3	$\boldsymbol{M}$	5.5×10^{-11}	8.19	2.92	2.80×	0.100	160
C3	$\boldsymbol{S}$	8.8×10^{-11}	81.92	24.27	3.37×	0.834	724
C3	$\boldsymbol{T}$	5.4×10^{-11}	2.05	1.69	1.21×	0.058	198
C3	$\bar{\boldsymbol{w}}$	–	0.03	0.03	1.00×	0.001	–
C4	$\boldsymbol{L}$	2.4×10^{-11}	0.46	0.26	1.76×	0.005	76
C4	$\boldsymbol{M}$	7.5×10^{-11}	41.47	4.70	8.82×	0.098	160
C4	$\boldsymbol{S}$	9.7×10^{-11}	414.72	40.39	10.27×	0.846	900
C4	$\boldsymbol{T}$	5.7×10^{-11}	4.61	2.33	1.98×	0.049	254
C4	$\bar{\boldsymbol{w}}$	–	0.06	0.06	1.00×	0.001	–
C5	$\boldsymbol{L}$	7×10^{-11}	1.04	0.33	3.14×	0.005	79
C5	$\boldsymbol{M}$	9.4×10^{-11}	209.95	6.45	32.55×	0.101	160
C5	$\boldsymbol{S}$	1.1×10^{-10}	2099.52	53.95	38.91×	0.843	969
C5	$\boldsymbol{T}$	7.3×10^{-11}	10.37	3.14	3.30×	0.049	315
C5	$\bar{\boldsymbol{w}}$	–	0.13	0.13	1.00×	0.002	–
C6	$\boldsymbol{L}$	2.3×10^{-11}	0.64	0.63	1.02×	0.032	71
C6	$\boldsymbol{M}$	8.4×10^{-11}	12.80	1.96	6.54×	0.100	106
C6	$\boldsymbol{S}$	6.3×10^{-11}	128.00	13.72	9.33×	0.702	206
C6	$\boldsymbol{T}$	5×10^{-11}	6.40	3.20	2.00×	0.164	166
C6	$\bar{\boldsymbol{w}}$	–	0.03	0.03	1.00×	0.002	–
C7	$\boldsymbol{L}$	5.4×10^{-11}	4.00	0.89	4.48×	0.003	66
C7	$\boldsymbol{M}$	7.9×10^{-11}	80.00	18.59	4.30×	0.056	309
C7	$\boldsymbol{S}$	6.4×10^{-11}	800.00	279.45	2.86×	0.837	546
C7	$\boldsymbol{T}$	5.5×10^{-11}	40.00	34.86	1.15×	0.104	337
C7	$\bar{\boldsymbol{w}}$	–	0.08	0.08	1.00×	0.000	–

Table C.9: E4 total construction time (seconds) per route and configuration. † marks a route that did not reach $\delta^\star$ within the 12 gigabyte cap; the time shown is the wall time to failure.

Config	Np	Uncompressed	Late	Mid	Early
C1	608	0.24	0.24	15	15
C2	1600	0.23	0.99	390	430
C3	3200	0.33	3.0	7700	7500
C4	7200	0.88	14	4300†	480†
C5	16200	5.7	70	1800†	1900†
C6	4000	0.77	6.0	490	450
C7	10000	16	100	770†	700†

Table C.10: E4 full transparency: total construction time, inverse-action solve time and its share of the total, peak resident memory, and status per route and configuration. The inverse-action solve dominates the time of every completed mid and early route.

Config	Route	total (s)	inv.-solve (s)	share (%)	peak RSS (GB)	status
C1	uncompressed	0.24	–	–	0.06	ok
C1	late	0.24	–	–	0.10	ok
C1	mid	15	15	96.3	0.51	ok
C1	early	15	14	96.1	0.49	ok
C2	uncompressed	0.23	–	–	0.11	ok
C2	late	0.99	–	–	0.49	ok
C2	mid	390	380	97.7	1.65	ok
C2	early	430	420	98.5	1.87	ok
C3	uncompressed	0.33	–	–	0.26	ok
C3	late	3.0	–	–	0.88	ok
C3	mid	7700	7600	99.9	5.20	ok
C3	early	7500	7500	99.8	5.67	ok
C4	uncompressed	0.88	–	–	0.69	ok
C4	late	14	–	–	2.63	ok
C4	mid	4300	–	–	13.21	fail
C4	early	480	–	–	13.03	fail
C5	uncompressed	5.7	–	–	2.88	ok
C5	late	70	–	–	9.03	ok
C5	mid	1800	–	–	8.53	fail
C5	early	1900	–	–	8.20	fail
C6	uncompressed	0.77	–	–	0.38	ok
C6	late	6.0	–	–	1.19	ok
C6	mid	490	480	96.8	3.84	ok
C6	early	450	440	97.0	4.16	ok
C7	uncompressed	16	–	–	1.69	ok
C7	late	100	–	–	4.06	ok
C7	mid	770	–	–	11.77	fail
C7	early	700	–	–	12.49	fail

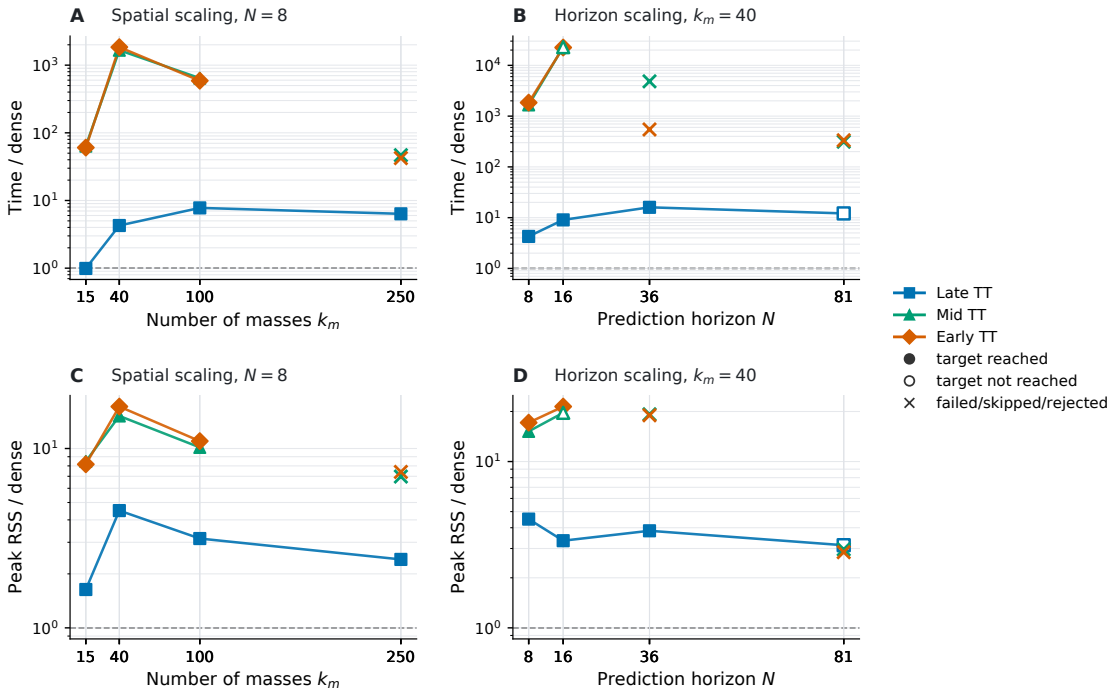


Figure C.4: E4 supporting: construction time (top) and peak resident memory (bottom) of the tensor routes relative to matrix-form construction. The late route stays within roughly five to ten times the matrix-form cost; the mid and early routes rise by orders of magnitude and fail (markers) on the largest cases.

C.7. Online latency (E5)

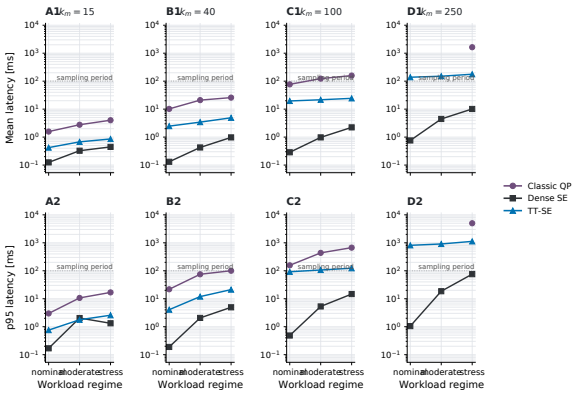
Table C.11 reports the mean per-step online latency of the online QP (OSQP), the matrix-form controller, and the tensor controller of section 5.9, with the tensor-to-matrix slowdown, as the median over the fifteen workload–seed runs of each configuration. The matrix-form controller is fastest in every case. The tensor controller is 2.2 to 53.7 times slower than in matrix form, with the slowdown smallest on the longest-horizon case C5, where the matrix-form controller has become memory-bound. Table C.12 resolves the latency by workload, giving the matrix and tensor mean, the tensor 95th-percentile and maximum, and the slowdown for each workload. The penalty grows with the constraint activity the workload drives, which is the active-set mechanism of section 5.9. Figure C.5 shows the same dependence across both sweeps.

Table C.11: E5 mean per-step online latency (milliseconds) and the tensor-to-matrix slowdown, median over workloads and seeds.

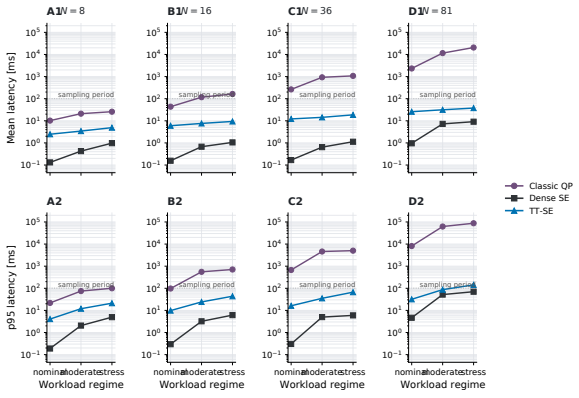
Config	Np	OSQP	Uncompressed	Tensor	Tensor/Uncompressed
C1	608	2.51	0.236	0.522	2.2×
C2	1600	18.5	0.366	3.53	9.7×
C3	3200	113	0.544	7.47	13.7×
C4	7200	956	0.817	14.2	17.3×
C5	16200	10100	6.76	31.3	4.6×
C6	4000	135	1.07	21.5	20.2×
C7	10000	1120	2.78	149	53.7×

Table C.12: E5 full transparency: matrix and tensor mean per-step latency (milliseconds), tensor 95th-percentile and maximum, and the tensor-to-matrix slowdown, per configuration and workload (median over the five seeds; maximum for the max column). The penalty grows with the constraint activity.

Config	Workload	Uncompressed mean	TT mean	TT p95	TT max	TT/Uncompressed
C1	nominal	0.114	0.425	0.678	1.61	3.7×
C1	moderate	0.355	0.665	1.21	5.24	1.9×
C1	stress	0.250	0.584	1.73	9.98	2.3×
C2	nominal	0.131	2.35	3.81	8.66	17.9×
C2	moderate	0.366	3.53	8.85	23.1	9.7×
C2	stress	0.778	4.50	14.7	36.1	5.8×
C3	nominal	0.125	6.07	7.35	15.1	48.5×
C3	moderate	0.544	7.47	18.5	29.9	13.7×
C3	stress	0.939	9.74	30.8	71.7	10.4×
C4	nominal	0.156	11.7	14.3	23.2	75.5×
C4	moderate	0.817	13.7	27.0	53.6	16.8×
C4	stress	0.968	18.7	48.0	81.6	19.3×
C5	nominal	0.946	25.1	29.6	43.6	26.6×
C5	moderate	6.76	31.3	64.8	122	4.6×
C5	stress	9.21	38.1	102	160	4.1×
C6	nominal	0.169	19.1	63.2	102	113.5×
C6	moderate	1.07	21.5	73.1	115	20.2×
C6	stress	2.18	24.4	84.2	145	11.2×
C7	nominal	0.733	139	545	828	190.1×
C7	moderate	2.78	149	605	968	53.7×
C7	stress	10.0	178	746	1210	17.8×



(a) Spatial sweep.



(b) Horizon sweep.

Figure C.5: E5 supporting: per-step latency by workload across the spatial and horizon sweeps. The penalty grows with the constraint activity the workload drives, consistent with the active-set mechanism of section 5.9.

C.8. Shape-plan variation (E6–E8)

Table C.13 gives the controlled shape-plan variation behind table 5.2 and the result of section 5.7. Starting from a single-temporal, single-physical baseline, one dimension at a time is split into smaller cores, and the final-operator parameter count and maximum TT rank are reported relative to that baseline. Splitting the temporal dimension (E6, the temporal split) barely moves the parameter count, keeping it within about 10% of the baseline at essentially unchanged rank. Splitting the physical dimension (E7, the physical split) is decisive. It cuts the parameter count to as little as 13% of the baseline while raising the maximum rank by up to a factor of nine. Storage and rank move in opposite directions under the split that helps, which can only happen because the total parameter count of equation (2.4), not the rank, sets the storage. The combined split (E8) confirms the pattern across all seven configurations.

Table C.13: Shape-plan variation: final-operator parameter count and maximum TT rank of a split shape plan relative to the single-temporal, single-physical baseline. Physical splitting cuts the parameter count while raising the rank; temporal splitting leaves both near the baseline.

Split (config)	Param. ratio	Rank ratio
Temporal split (E6), horizon sweep		
C2	1.004	1.00
C3	1.059	1.06
C4	1.045	1.06
C5	0.912	1.07
Physical/dynamics split (E7), spatial sweep		
C1	0.367	1.48
C2	0.697	8.63
C6	0.128	3.13
C7	0.368	8.52
Both split (E8), all configurations		
C1	0.375	1.50
C2	0.710	8.81
C3	0.748	7.33
C4	0.779	6.07
C5	0.689	5.37
C6	0.131	3.21
C7	0.376	8.71